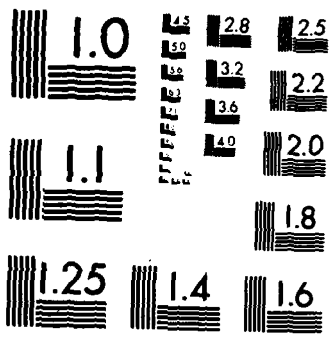
AD-A136 835 A SPOKEN ENGLISH RECOGNITION EXPERT SYSTEM(U) AIR FORCE 1/3
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING R L ROUTH SEP 83 AFIT/GCS/EE/83S-01

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A136835

AIR FORCE INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY
UNITED STATES AIR FORCE

A SPOKEN ENGLISH RECOGNITION
EXPERT SYSTEM

by
Richard LeRoy Routh, B.S., M.A.M.
Captain, Signal Corps, U.S. Army

Graduate Computer Systems
September 1983

DTIC FILE COPY

SCHOOL OF ENGINEERING

DTIC ELECTE JAN 12 1984 S B

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

83 12 13 240

(1)

A SPOKEN ENGLISH RECOGNITION
EXPERT SYSTEM

by
Richard LeRoy Routh, B.S., M.A.M.
Captain, Signal Corps, U.S. Army

Graduate Computer Systems
September 1983

AFIT/GCS/EE/83S-01

A SPOKEN ENGLISH RECOGNITION
EXPERT SYSTEM

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Richard LeRoy Routh, B.S., M.A.M.

Captain, Signal Corps, U.S. Army

Graduate Computer Systems

September 1983

Approved for public release; distribution unlimited

~~U.S. Government Agencies~~

Preface

The advent of a computer which can intelligently communicate with a human being in his native language will so revolutionize human society that a 50 bit per second machine cannot at this time begin to grasp the breadth and depth of the change it will bring.

There are few, if any, research efforts that can offer the exhilaration and challenge of exploring how the human brain processes speech. Of the means of communicating in the animal kingdom, only speech is unique to humankind. Perhaps that is an evidence of its power and complexity.

There are few people who have been as privileged as I to study under the brilliance and enthusiasm of Dr. Matthew Kabrisky. There are even fewer who have had the very unique opportunity to take full advantage of the very important and timely work of Dr. Robert W. Milne. For these opportunities, I am truly grateful.

I am very thankful to my wife, Edie, who provided support in every way possible. Without her contributions, this work would not have been realized.

Richard LeRoy Routh
WPAFB, OH
August 1983

Contents

Contents

Delete in distribution statement: for U.S. Government Agencies per Capt. Milne, AFIT/EN

List of Figures

Abstract

Subject to the accuracy of the acoustic analyzer and the accuracy and completeness of the English Parser, a real-time general solution to the application of English syntactic constraints to spoken English recognition has been developed. This solution is functionally equivalent, in many ways, to the syntax processing of spoken English in the human brain. Because it closely models the syntax processing of the Human Speech Recognition System (HSRS), it is most effective when used with the several levels of semantic analysis which are also evidently operational in the HSRS as has been shown in this thesis. Hence, this work may well be a necessary part of the the eventual general solution to the English speech recognition problem.

I. Introduction

As computers become more capable of performing complex and intelligent tasks, they become both more useful and easier to use. Computers are becoming more useful because their capabilities to solve complex problems is increasing. They are becoming easier to use because their increased speed, size, and complexity allow them to be programmed to use the communication methods which humans prefer. The more this happens, the less special training is needed on the part of the user. One of the goals of this evolution is to provide the layman with the capabilities which the computer can afford him without any special training whatsoever. The ability to type, the learning of computer protocols, and the speaking with distinct and separated speech all fall into the category of special training. There is, therefore, a need to build an interface to computers which is capable of understanding normal speech. This problem is referred to as the man-computer communications gap.

This communications gap between humans and computers is evident in that only a small fraction of the human population is sufficiently educated to be able to use computer systems. A significant portion of bridging this gap involves the learning of special codes, languages, typing skills, et cetera, in order to train the human user to communicate with the computer.

The task of providing these problem solving capabilities to a great many more people would be much more easily realized if the computer could be taught to understand common English rather than people being taught to communicate in computer languages. It is to this end that this research is directed.

A great deal of researc. has already been directed at shrinking the man-computer communications gap. Other than for some extremely restricted applications, this research has produced small vocabulary, speaker dependant, isolated word speech recognizers. What is required is a real time, large (virtually unlimited) vocabulary, speaker independant, connected word speech recognizer.

A normal spoken English interface to a computer promises to provide the enormous capabilities of the electronic computer to the non-technically trained person. This would facilitate great technical advances in all fields of human interest and endeavor. A speech interface with a computer is a difficult problem to solve. Its scope and complexity are beyond an individual masters degree level research effort. However, since complex problems are often solved by dividing them into less complex subtasks and then solving these simpler subtasks one at a time, this researcher proposes to approach the solution to this difficult problem by solving one of its subtasks.

A. Background and Problem

In order to provide a solution to the common speech computer interface problem as previously described one should consider the following model:

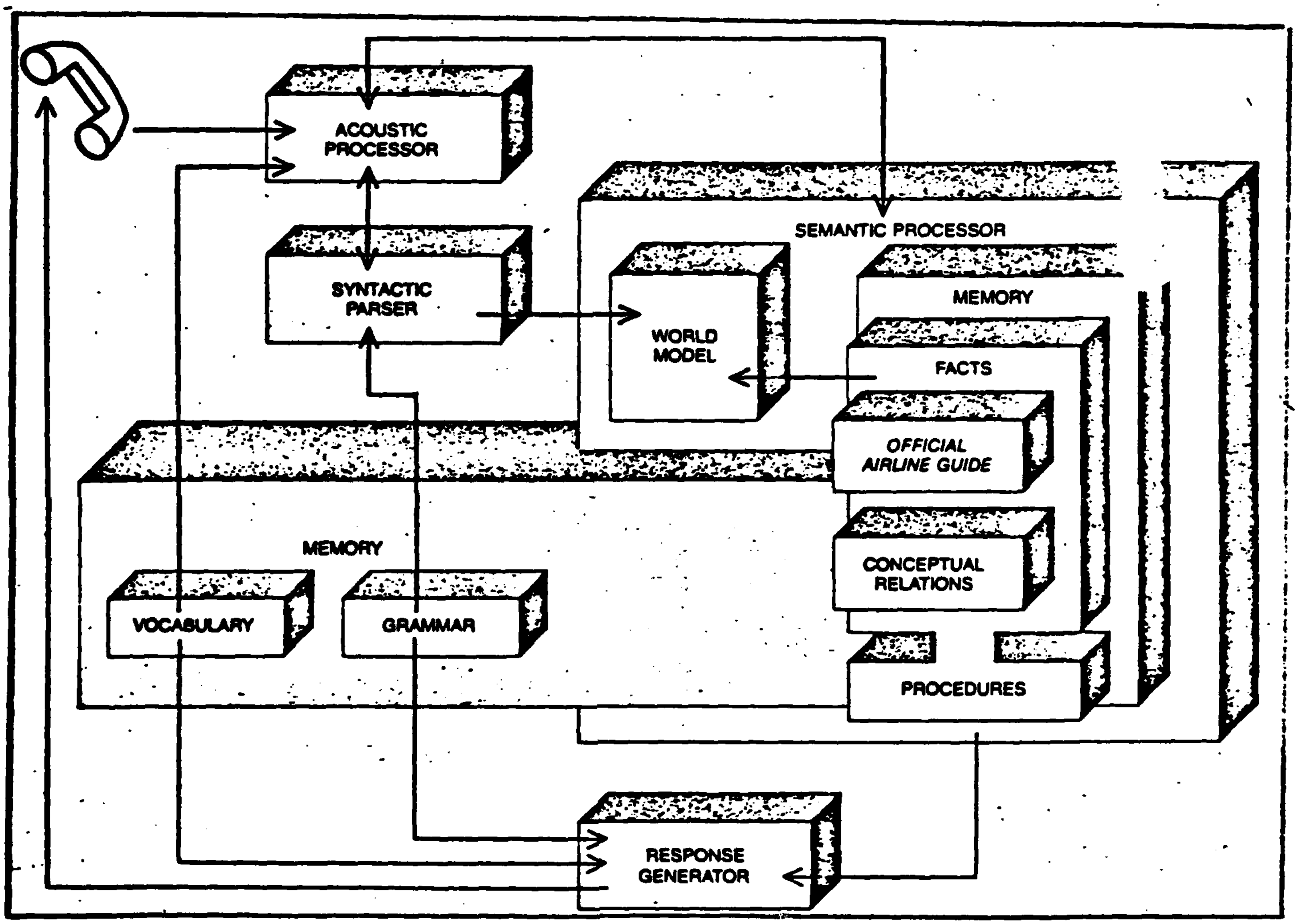


Figure 1.1. Hierarchies of Speech Recognition

This model was proposed by Levinson (Ref 15:76). It is believed to be an accurate psychological model of how speech is processed in the human brain and will be discussed in more detail in later chapters. It can be seen that in this approach the first step is to process the output of the voice decoder (which is the acoustic processor in Figure 1.1) through a syntactic parser. This is necessary because contemporary voice decoders are not accurate enough, nor is there sufficient information in word sounds alone, to correctly reconstruct the input speech. Both syntactic and semantic feedback should be provided to the voice decoder in order for it to choose properly from among its available decoded options.

For example: In the sentence: "The ewe had a lamb," the word "ewe" sounds identical to the word "you" and the pronunciation of the letter "u." It is not possible for the voice decoder to consistently make the right decision based only on the sound of the word. It is often found (especially in connected speech) that both syntactic information (grammatical correctness -- which is sufficient in this example) and semantic information (meaning) are needed in addition to the accurate mapping of the sound of the uttered input to the sounds of the words in the computer's vocabulary.

The purpose of this research and thesis is to build such an interface between the voice decoder being developed here at the Air Force Institute of Technology under the guidance of Dr. Matthew Kabrisky and Major Larry Kizer, and the Syntactic/Semantic English parser which was developed by Dr. Robert Milne at the University of Edinburgh, Scotland.

B. Scope of Solution

It is important to realize that a complete solution to the Spoken English Recognition Expert System (SPEREXSYS) problem is a ongoing process. The quality of the solution is dependant on the quality of the voice decoder and the quality of syntactic and semantic analyzers. The quality of the voice decoder is dependant on its accuracy, its vocabulary, how well it handles connected speech, how well it understands various dialects, its look back ability, and so forth. The quality of the syntactic and semantic analyzers depend on their vocabularies, the completeness and correctness of their grammars and semantic rules, the accuracy of the algorithms they use to decide among otherwise equally viable options, and so forth. It is therefore apparent at the outset that the scope of the SPEREXSYS solution is greatly constrained by the quality of the two modules it interfaces.

The interface between the voice decoder and the English parser is useful in that it promises to improve the accuracy of the voice decoder by assisting it in choosing among otherwise nearly indistinguishable decoding alternatives. It does this by selecting the highest probability voice decoder outputs and allowing the English parser to comment on their grammatical correctness. The precise algorithms for doing this will be described in chapter three. The SPEREXSYS then selectively applies these comments to the voice decoder output and feeds the results back to both the voice decoder and the English parser until an acceptable solution has been found. An acceptable solution is defined to be one which is grammatically correct and which is above the error threshold of the voice decoder. It follows then that a major purpose of this project is to determine how much of an impact the English parser has on the reliability of the voice decoder output.

In addition, the SPEREXSYS is to be written and documented in such a way that it lends itself easily to modification in order to incorporate improvements in not only its own software but also future improvements in the voice decoder and the English parser.

C. Assumptions

This thesis effort is predicated on three assumptions. The first is that the voice decoder is fairly accurate. This means that the word which was actually uttered into the voice decoder's input will appear in the top few choices of the voice decoder's output.

The second assumption is that the English parser accurately analyzes the grammar of the candidate sentence strings which are input to it. This includes the requirement to assess the degrees of grammatical correctness (see the discussion from Bach in the next chapter) of the candidate sentence strings.

The third underlying assumption is that a two hundred word vocabulary is large enough to demonstrate the feasibility of this approach. Fifty word vocabularies appear to be an upper limit of commercially available voice decoders. If the approach in this thesis can be shown to work for at least two hundred word vocabularies, then it will be successful in demonstrating both the philosophy and methodology of this thesis approach because it will have improved the state-of-the-art performance by applying syntactic constraints to the output of the voice decoder. Two hundred words is thought to be large enough to demonstrate the success of this approach while, at the same time, being small enough to work with in the limited time constraints of an AFIT masters degree thesis.

D. General Approach and Summary of Current Knowledge

The general approach to solving this problem is as follows:

The interface accepts all outputs from the voice decoder. The voice decoder outputs will be all the words from its vocabulary which have matching scores above some previously defined error threshold. This will be explained in greater detail in chapter two. These outputs comprise the voice decoder's best guesses of what was uttered. The SPEREXSYS then strings these best guesses together based on the time sequences of reception into the voice decoder. The most probable strings are then sent to the English parser for analysis. The parser determines whether or not the strings it has been sent are grammatically correct. If they are grammatically correct, then it signals the semantic levels of the SPEREXSYS that an acceptable solution has been found. If it is not grammatically correct, then the SPEREXSYS eliminates that string from further consideration. Since the SPEREXSYS builds grammatically correct strings deterministically (one word at a time), several grammatically correct high probability sentences are constructed from a single uttered sentence. The syntactic levels of the SPEREXSYS appeal to the semantic levels for arbitration of these ambiguities.

To this researcher's knowledge, no such interface has ever been attempted. This may be due in part to the fact that Dr. Milne's English parser has only recently been completed and is the only accurate (psychologically correct model of the human speech recognition process) English parser which allows the deterministic parsing of a sentence. It will be shown later in this thesis that this characteristic is essential to the successful building of an interface between a voice decoder and an English parser. Many decisions on error thresholds have had to be made for the first time. Many algorithms on option selections (and the associated selection criteria) have been developed as original work. Many decisions in these areas have been made for the first time because up until this thesis, this was an unsolved problem.

E. Standards

If, within a few seconds or less (near real time), the SPEREXSYS can successfully pick the correct string (based on ambiguities which only need syntactic constraints for correct decisions) from among the millions of possible strings which can be constructed from its top few choices at each word of sentence lengths of about ten words (if only the top five choices for each word of a ten word sentence are used to construct candidate strings, there are 9,765,625 possible strings that can be constructed), and if it can do so repeatedly for different uttered sentences, then the concept and methodology used to build the SPEREXSYS will be considered validated.

F. Materials and Equipment

The materials and equipment which were needed to complete this thesis were all available at the outset of the thesis. They are listed as follows:

1. VAX 11/780 computer and the Franz LISP compiler and interpreter,

2. The English parser and the Avionics Laboratory DEC-10 computer on which it runs,

3. The voice decoder and the Pattern Recognition Laboratory computer on which it runs, and

4. Four modems and their associated computer ports which will be used to connect the VAX computer with the DEC-10 computer and the VAX computer with the Pattern Recognition Laboratory computer.

G. Other Support

Computer center operations personnel for all three computers were required to identify modem ports and to hook up the modems which will connect the computers together.

H. Sequence of Presentation

This first chapter provides the reader with a broad perspective of where this research fits in the world of computer interface developments. It serves to acquaint the reader with the background relevant to this research and to provide a brief description of the purpose, scope and complexity of the research which was done for this thesis.

The second chapter has been written to provide the reader with a more detailed understanding of the problem. Included in this chapter is a discussion of the theory of speech recognition and how the voice decoder implements that theory, as well as a discussion of the inadequacies of a stand-alone speech recognizer.

Also included in the second chapter is a discussion of transformational grammar and the implementation of this grammar theory into the English parser used in this research. The chapter concludes with a discussion of the concepts which relate to the solution of this problem.

Chapter three describes the structure and design of the Spoken English Recognition Expert System (SPEREXSYS). The design is presented in three phases. Phase one describes the top level design which discusses the system interfaces and the reasons for choosing them in the manner used.

The second phase explains the intermediate levels of design through the use of structure charts. Design problems are discussed and the rationales used in making the design decisions are described.

Lastly, this chapter briefly discusses the low levels of the design.

Chapter four deals with the specific implementation of the design as it relates to the machine peculiar interfaces. Other miscellaneous implementation details are discussed. The rest of the chapter is devoted to explaining the validation and testing philosophy and procedures. Each test is defined in terms of how it was used to help validate the design. The results of each test are discussed as well as an explanation of the conclusions which are drawn from the analysis of the test results.

The fifth and final chapter presents a summary of this thesis, an explanation of how the SPEREXSYS can and should be used, and a discussion of the recommended improvements and enhancements to the system. The chapter concludes with a presentation of the possible future extensions of this work.

Appendix A is a listing of the Franz LISP code of the SPEREXSYS. This listing includes many comments on the function of the code, line-by-line, and module-by-module.

Appendix B contains the results of selected sample runs from the testing.

Appendix C is a short users manual which should be of great help to future SPEREXSYS users and developers.

Appendix D is a discussion of how what is already known about the Human Speech Recognition System (HSRS) can and should be more reflective of speech recognition systems as they have been developed to date.

Appendix E discusses the function of a short term memory and how it relates to speech recognition. A brief example is included.

Appendix F presents the formulation and experimental verification of the hypothesis that the HSRS favors longer words over shorter words.

Appendix G is a brief data dictionary which includes primarily global data descriptions. A few key local data descriptions are also defined.

II. Existing Framework

The nature of the SPEREXSYS program is to function as a smart interface between the Voice Decoder and the English Parser. It is thgrefore necessary to understand the theory and the function of both the Decoder and the Parser in order to gain an appreciation for the parameters which constrained the potential performance of the SPEREXSYS from the outset of its development. This background will also assist the reader in understanding why certain SPEREXSYS design decisions were made.

This chapter discusses the theory and function of both the Voice Decoder and the English Parser. After this background foundation has been laid, the chapter will conclude with an explanation of the concept of the solution to the SPEREXSYS design problem.

A. The Voice Decoder

The Air Force Institute of Technology has been developing a speech recognition machine for the past few years under the guidance of Dr. Matthew Kabrisky and Major Larry Kizer. Its incremental development and improvement has been the result of the efforts of a series of students, each working on a graduate level research project. A conceptual summary of their combined research and results is presented here in order to acquaint the reader with a general understanding of the theory and function of the Voice Decoder used in the development of the SPEREXSYS. In addition, some general theory of the acoustic analysis of speech is discussed along with mentions of other approaches to solving the same problems.

Referencing the Levinson model (Figure 1.1), the Voice Decoder performs the function of the acoustic processor. Specifically, it examines the spoken input utterance and makes guesses as to what words might have been spoken. Doing only this much has consumed the efforts of some of science's brightest people for over two decades without completely satisfactory results to date. By "completely satisfactory" it is meant that the acoustic processor (voice decoder) approaches the accuracy of the acoustic analyzer in the human speech recognition system (HSRS). As this discussion develops, the reader should remember two things:

1. Acoustic processors have been developed to the point that they are moderately accurate at guessing words from a controlled input string when the vocabulary of possible choices is restricted to less than 100 words.
2. A completely satisfactory acoustic processor is only a small (nevertheless critical) functional subset of the speech recognition

process (see Figure 1.1). The upper levels of the process -- the syntactic, semantic, and response generation levels -- are dependent on a reliable "front end." That is to say: a chain is only as strong as its weakest link and a speech recognition system is no exception.

The spoken input is in the amplitude versus time domain. This is the form of the output of a microphone. It is also the form of the output of the ear drum to the inner ear mechanism. Since, as has already been mentioned, sounds vary so considerably when spoken by a human speaker (even when he attempts to reproduce the same sound), simple direct template matching of stored sounds to the output mapping of a sound in the amplitude versus time domain is woefully inadequate. There are amplitude and frequency variations as well as time warping between any two sounds produced by a human so as to make straight template matching extremely unreliable. There are word recognition schemes which attempt to normalize amplitudes, allow for small variations in frequency, and employ complex time warping. This approach has consistently proven to yield unreliable results and to be extremely computation bound for a general solution. This tends to suggest that a different feature set must be examined.

An examination of the process in the HSRS reveals that the signals are converted from the amplitude versus time domain (output by the ear drum) to the amplitude versus frequency domain (the signal in the auditory nerve) where the frequency axis is logarithmic. Many approaches incorporate this knowledge by attempting to classify the spoken inputs on the basis of some form of a Fourier transform feature set. The AFIT Voice Decoder employs this method of analysis in order to classify eight millisecond time slices of uttered speech as specific phonemes.

Some speech recognition approaches such as Linear Predictive Coding (LPC) do not use a Fourier based feature set, but instead, attempt to classify the uttered input based on feature sets which are probably significantly different than those used by the HSRS.

Some acoustic analyzers do use Fourier based feature sets, but process entire words instead of first breaking these words into phonemes. Of course, before this can be done, the beginning and end of words must be known. This is not so hard if the words are spoken so that there are significant time gaps between words and if the ambient noise is low. This (time gaps between words) is called discrete speech or isolated word recognition. While this is not the way English is naturally spoken, it is, however, a constraint which greatly simplifies speech recognition.

The AFIT Voice Decoder currently uses isolated word recognition. It is extendable to connected (natural) word speech but this has not yet been done. The reason it can be extended to connected speech is that its analysis of word choices is based on a serial string of phonemes which represent the uttered input. This greatly constrains the possible word boundaries to such a small set that exhaustive searches of word boundaries becomes feasible.

The speech recognition methods which do not first identify phonemes before attempting to identify words must rely on some type of time warping algorithm. This is necessary because a word can be compressed and expanded at multiple points (in time) in its utterance when compared to a previous utterance of the same word. For example, one utterance of the word "three" may take 250 milliseconds to produce. Even if all future utterances of the word "three" are time normalized to 250 milliseconds, the duration of the "th" sound may be 40 milliseconds for one of the utterances and 50 milliseconds for the other. Similar differences for the other phonemes in "three" are also likely. So until these time warpings of the different phonetic sounds in a word are time warped back to match the standard representation of the word (the store prototype against which all future utterances will be compared), no significantly reliable mapping of an input utterance to a stored prototype can occur.

In the AFIT voice decoder, once a string of phonemes has been output from the first stage of the Voice Decoder (Ref 20), the second stage analyzes this string to determine the best match of this string of phonemes with the phoneme strings which are characteristic of the words in its vocabulary (Ref 18). This is a very difficult and complicated problem beause the output strings are quite variable.

The difficulty lies in the fact that, because the input speech is quite variable, the first stage (phoneme identification) characteristically makes many wrong guesses. The second stage, therefore, must attempt to find a best word match using inaccurate input. To do this, straight template matching has proven not to work well.

To illustrate the problem, refer to figure C.6 (Seelandt:260). Figure C.6 is reproduced here as figure 2.1. It shows the output of the first stage of the Voice Decoder for 48 time slices. This output for the utterance "zero" is in the form of a table of the best five guesses (with weighted degree of certainties normalized to 100) for the input utterance for each time slice. The following questions/problems are immediately apparent:

1) When does the word start and stop?
2) When should one discard data due to excessive background noise or transition between phonemes?
3) What algorithm, if any, should be used to decide which of the five most probable guesses to use for each 8 millisecond time slice.

200	XX-100	XX- 1	OH- 49	OU- 39	ZX- 34
201	XX-100	XX- 70	OH- 48	OU- 38	ZX- 34
202	XX-100	XX- 71	OH- 49	OU- 38	ZX- 34
203	XX-100	XX- 67	[illegible]- 42	[illegible]- 37	[illegible]- 32
204	XX-100	XX- 66	ZX- 46	RE- 44	OH- 33
205	XX-100	IE- 63	XX- 63	RE- 62	ZX- 57
206	IE-100	XX- 98	IE- 90	KX- 89	TX- 86
207	IE-100	RE- 97	IE- 97	AE- 94	EE- 93
208	IE-100	EE- 96	RE- 91	EE- 88	IE- 88
209	IE-100	EE- 94	FX- 90	IE- 90	FX- 89
210	IE-100	FX- 97	RE- 96	EE- 96	EE- 96
211	RE-100	OU- 96	IE- 94	FX- 93	IE- 93
212	ZX-100	OU- 94	RE- 94	AY- 90	NX- 84
213	RE-100	ZX- 96	OU- 95	AY- 92	NX- 85
214	ZX-100	AY- 98	IY- 97	NX- 95	OU- 91
215	IY-100	AY- 99	ZX- 98	OU- 92	IY- 90
216	ZX-100	IY- 98	AY- 95	ER- 89	RE- 89
217	ZX-100	AY- 94	NX- 92	IY- 91	IY- 90
218	ZX-100	RE- 97	NX- 96	ER- 95	IY- 95
219	RE-100	ER- 96	OU- 96	ZX- 94	NX- 94
220	OU-100	RE- 94	ER- 89	NX- 89	NX- 88
221	OU-100	UA- 95	RX- 88	ER- 87	NX- 86
222	OU-100	UA- 99	VX- 96	ER- 87	RE- 85
223	UA-100	VX- 92	OU- 89	RA- 82	ER- 79
224	UA-100	RA- 93	OU- 93	VX- 91	OR- 81
225	OU-100	UA- 96	RA- 94	OH- 93	VX- 90
226	OU-100	UA- 98	TX- 89	OH- 88	OR- 84
227	OU-100	UA- 97	OR- 95	OH- 90	OU- 85
228	UA-100	OU- 98	OU- 86	NX- 84	OR- 83
229	UA-100	OU- 95	OU- 88	VX- 87	WU- 85
230	OU-100	WU- 98	UA- 95	OU- 92	NX- 92
231	OH-100	OU- 99	RA- 96	WU- 95	UA- 95
232	OU-100	OU- 97	UA- 96	RA- 95	TX- 90
233	OU-100	OU- 82	WU- 81	NX- 79	UA- 78
234	OU-100	OU- 86	NX- 80	WU- 78	UA- 77
235	OU-100	WU- 84	OU- 80	NX- 80	UA- 77
236	OU-100	WU- 91	OH- 82	NX- 79	OU- 74
237	OU-100	WU- 88	OH- 86	NX- 81	OH- 72
238	OU-100	WU- 90	OH- 90	NX- 82	OH- 73
239	OU-100	OH- 97	WU- 94	OH- 81	NX- 77
240	OH-100	WU- 90	OU- 89	OH- 82	WU- 77
241	OH-100	WU- 91	OH- 87	RA- 86	OH- 84
242	OH-100	PA- 92	OH- 91	WU- 89	WU- 88
243	OH-100	OH- 99	WU- 94	OR- 90	RA- 89
244	OH-100	RA- 89	OR- 89	WA- 89	OH- 87
245	OH-100	OH- 99	RA- 95	WA- 94	RX- 92
246	R[illegible]-100	AH- 97	OH- 96	WA- 95	RA- 92
247	AH-100	WA- 94	OR- 98	RA- 86	RX- 84

Figure 2.1. Example output (OUT2) of measurement routines (TRYDIST5 and LISTER2) using second (Hamming window) prototype set on the utterance "zero".

To solve these problems in the second stage of the Voice Decoder, Fuzzy Set Theory was used. Fuzzy Set Theory allows for partial membership in a set. Each input phoneme has some degree of membership in all phoneme sets (templates) stored in the program's dictionary. Its degree of membership is determined by the likeness of its (the input phoneme) elements to the elements in the stored-dictionary phoneme's set.

The Fuzzy Set Word Guessing algorithm identifies characteristics for comparison as elements within the sets and computes coefficients of likeness between the input set elements and the stored word set elements. These coefficients are each weighted according to the program's algorithm to determine the input string's degree of membership in the sets which represent the stored words. The stored word set in which the input string has the highest membership is the first and best guess as to the identity of the word which was input into the Voice Decoder. The second guess word is the word set in which the input string has the next greatest membership and so on.

The coefficients of the elements of each set are tuned dynamically and heuristically by both the programmer at initialization and the program during execution. This allows for the continued improvement of accuracy of the program.

It has been mentioned that isolated word speech is easier to recognize than connected word speech. That is because in isolated word speech the word boundaries are clearly defined. In natural (connected) speech they are not clearly defined. It is common in connected speech to have the initial and trailing phoneme of a word be somewhat mutilated due to the fact that the trailing phoneme of a word is required to slide evenly (acoustically transition) into the initial phoneme of the next word. In essence, the two compromise slightly in order to transition smoothly. This results in phoneme mutilation and, therefore, makes the task of phoneme identification more difficult. In addition to this complicating phenomenon, when a word's trailing phoneme and the next word's initial phoneme are the same, they are commonly shared. For example, in the normal utterance of the two words "white towel," the phoneme "t" occurs only once. It is shared by both the word "white" and the word "towel." Sometimes even more than one phoneme is shared when transitioning from one word to the next. This more commonly happens when the speaker is speaking rapidly. The two words "mast string" when spoken quickly may share both the "s" and the "t" sounds. In this instance, these two words are also acoustically indistinguishable from the single word "mastering." In order to find the word boundaries in connected speech, since acoustic analysis is insufficient in determining them (even with a perfect acoustic analyzer, there is insufficient information in the acoustics alone to do this), syntactic and semantic information must be used. That is one of the reasons that this thesis has been performed. It allows only grammatically correct (syntax) word strings to be formed. This syntactical constraining of word guessing in the acoustic processor helps to find word boundaries (and improve the reliability of word guesses) but even perfect syntactical analysis is insufficient to resolve many ambiguities. The following example helps to illustrate that several levels of semantics are also required in order to accurately recognize conversational speech:

> Mary works in the cosmetic section of a large department store. She has just completed the monthly inventory and is now engaged in ordering the items which are in short supply. Her boss inquires as to whether she is going to be ordering both hand lotions and facial makeup kits. She replies, "I am ordering more hand lotions because we're pretty low on them this month. We still have a pretty good supply of makeups though, so I won't be doing any makeup ordering."

If we input only the last two words of Mary's reply to a Voice Decoder which is capable of perfect accuracy in interpreting input phonemes (which, of course, AFIT's is not at the present time), the following interpretations would be completely legitimate:

1) makeup ordering

2) make up ordering

3) make cup ordering

4) may cup ordering

5) makeup or during

6) makeup order ring

7) makeup poor during

8) makeup portering

9) makewp porter ring

10) make up or during

11) make up order ring

12) make up poor during

13) make up portering

14) make up porter ring

15) make cup or during

16) make cup order ring

17) make cup poor during

18) make cup portering

19) make cup porter ring

20) may cup or during

21) may cup order ring

22) may cup poor during

23) may cup portering

24) may cup porter ring

25) make a poor during

26) make a porter ring

27) make a portering

Of these 27 distinct phoneme based interpretations of the two words which were input into the hypothetically flawless Voice Decoder, only the first, second, third, fourth, eight, thirteenth, eighteenth, and thenty-third can fit syntactically with the rest of the sentence. Only the first four of those fit semantically within the context of the sentence; and only the first one (makeup ordering) fits semantically within the context of the entire conversation.

This example illustrates some of the problems encountered when attempting to interpret the utterances of connected speech with even a perfectly accurate Voice Decoder. It also illustrates the ultimate need to filter the Voice Decoder's output through first a syntactic analyzer, then a sentence-contextual semantic analyzer, and finally through a global-conversation-contextual semantic analyzer.

The last performance criteria of an acoustic processor which will be discussed in this chapter is the vocabulary size. As the accuracy of an acoustic processor increases, the distinguishability between the words in its vocabulary increases. As this distinguishability between words (resolution) increases, the vocabulary size which can be reliably (for some given degree of reliability) differentiated increases. The size of the vocabulary which a voice decoder can handle is therefore limited by its accuracy for some given degree of reliability. Since the addition of syntactic and semantic constraints on the word guessing of the input improves the reliability of a speech recognizer, it also allows for an increased vocabulary size. It should be realized, however, that if the vocabulary search is not done in parallel, the processing time will be increased for larger vocabularies.

The astute observer will realize that the accuracy of the state-of-the-art speech recognizers is fairly poor as evidenced by the fact that they all restrict the upper bound of the vocabulary size to less than one hundred words. It is this researcher's contention that the solution to this accuracy problem is to more closely mimic the functions of the HSRS. Some suggestions for this are contained in chapter five and appendix D of this thesis.

B. The English Parser - Its Theory and Application

The English Parser is the tool which the SPEREXSYS used to insure that only interpretations which are grammatically correct were accepted. The English Parser is the result of the Ph.D. research done by Robert W. Milne. It continues to undergo modification and improvement as new rules and requirements are identified (especially during the development of the SPEREXSYS). Because the English Parser was structured in a strictly top-down modular fashion, it lends itself easily to expansion and modification. Before examining the specific theory and design of the English Parser and its functional application as a grammatical filter in this project, it is useful to discuss the nature of a grammar. This discussion will lead to an understanding of both why a transformational grammar was used and why the particular architecture of Milne's Parser is ideally suited for use in a project such as this one.

Koutsoudas has said that "a grammar is a device that tells the reader [user] how to construct an infinite number of correct sentences of a language and no incorrect ones (Ref 13:1)." In order to develop a good computer program which could be used as a syntactic filter, it was necessary to study English grammar and then to build a program which was an accurate model of the rules of that grammar.

The task of choosing a good English grammar to use as a syntactic filter for common speech is complicated by the observed phenomenon in English that there exist varying degrees of grammatical correctness. Koutsoudas points out that there are "maximally grammatical" and "questionably grammatical" sentences in English. A good grammar should generate the maximally grammatical sentences first in order to be able to identify the deviations of the questionably grammatical sentences (Ref 13:2). One must be careful, therefore, to choose not only a grammar which generates only correct English sentences, but one which preferentially generates the most correct ones.

There are two approaches to formulalting a grammar. These are the Familiar Linguistic Theory approach and the more recent Transformational Analysis approach. In comparing these two approaches, Chomsky writes:

> Our main conclusion will be that familiar linguistic theory has only a limited adequacy - i.e., that it is attempting to do too much with too little theoretical equipment. ...It will be shown that the theory of transformational analysis can be formulated in the same completely distributed terms that are required anyway for lower levels and that a large and important class of problems that arise in the rigorous application of familiar linguistic theory disappears when it is

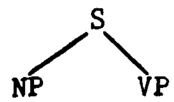
> extended to include transformational analysis (Ref 5:64).

Familiar Linguistic Theory approaches the task of formulating a grammar by listing a separate rule for each specific case of sentence formulation. This approach produces grammars which are extremely lengthy because each of its rules specifies an extremely restricted class of sentences. One might well argue that Familiar Linguistic Theory is only a very large collection of trivial cases. On the other hand, Transformational Analysis attempts to describe a grammar in terms of General Linguistic Theory. Bach elaborates on the nature of General Linguistic Theory:

> General Linguistic Theory ... must present a set of terms and distinctions sufficient to account for the rich variety of grammatical systems given in the world's several thousand languages, but limited enough to explain the universal features of these natural languages. Each theory of a specific language can then be taken as a particular exemplification of the types of systems predicted by General Linguistic Theory. To the extent that the notions of Transformational Theory are adequate to this task, it offers a preliminary picture of what languages in general are like (Ref 2:2).

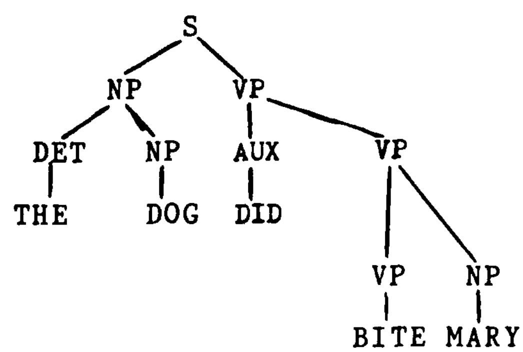
The following brief summary and examples are offered to acquaint the reader with the general concept of an English Transformational grammar.

Transformational Grammar begins with the hypothesis that every sentence is composed of two structural elements: a noun phrase and a verb phrase. Graphically this concept is presented as follows:



From this hypothesis, a complete English grammar can be generated using less than thirty rules. (This is a major reduction from the hundreds of rules required by Familiar Linguistic Theory to specify only a subset of English grammar).

For example, one rule of Transformational Grammar is that the noun phrase can be implied. Hence, the sentence, "Go!" has an implied noun phrase in the second person and the verb "Go" is the complete verb phrase. In the sentence, "The dog did bite Mary," the noun phrase and verb phrase are both expanded according to other Transformational Grammar rules:



There are two types of rules in Transformational Grammar. They are P-rules and T-rules. P-rules are rules which allow phrase replacement for single elements. T-rules are Transformational rules which allow for the transformation of sentential elements to reconstruct different, but legal, sentences from a root sentence (Ref 13: chapter one).

It is a P-rule which allows a noun phrase to be replaced by a determiner and a noun phrase. (Hence "the dog" satisfies the structural requirement for a noun phrase in the original hypothesis).

The concept of using transformations such as the passive and question transformations make for a grammar that avoids the need for "rules of incredible complexity" which the more traditional (Familiar Linguistic Theory) approaches require (Ref 1:100-101). The following example helps to illustrate this.

In the following sentence: "Did the dog bite Mary?" it is a T-rule which allows the statement to be transformed into a question by simply changing the position of the auxilliary from the third word to the first word in the sentence. Another rule specifies that all sentences have auxilliaries even though some are understood and do not appear in the original sentence. Hence the sentence, "The dog bit Mary" can also be transformed into the question: "Did the dog bite Mary" (by use of the previously demonstrated T-rule and another rule which changes the form of the verb from "bit" to "bite") or into the question: "Has the dog bit Mary?"

The general philosophy of Transformational Grammar is embodied in a statement from Chomsky:

> In general, we introduce an element or a sentence form transformationally only when by so doing we manage to eliminate special restrictions from the grammar, and to incorporate many special cases into a single generalization (Ref 5:416).

Transformational grammar, therefore, is a general theory of grammar which provides general rules for describing legal sentence syntax. For example, it is this transformational grammar which allows one to say: "The boy runs," but does not allow: "The boys runs." It specifies that in the second case, the "s" must be dropped off the end of the verb "run" in order to produce a syntactically (grammatically) legal sentence. It is also transformational grammar which specify proper word order so that "The red ball" is syntactically legal, but "The ball red" and "Ball red the" are illegal syntactic constructions. These sorts of rules form the basis for the application of the syntactic constraints which help in the problem of identifying illegal word combinations such as those likely to be produced by the acoustic analyzer.

A good English parser must be a functional duplicate (produce identical outputs for a given set of inputs) of the Human Sentence Parsing Mechanism (HSPM), that is to say, it must fail where the HSPM fails and it must succeed (display relative computational speeds for different types of parsing problems) where the HSPM succeeds (Ref 17: chapter one). This is accomplished in Milne's parser by the techniques of "limited lookahead" and "wait and see." "Limited lookahead" means looking ahead in the input stream before deciding which grammar rule to execute and hence, which will be the next state (Ref 17:16). "Wait and see" means that, if the parser is unsure of a situation, it does not make a random guess. Instead it waits until it has enough information to make the decision correctly (Ref 17:16). By employing these two techniques in conjunction with transformational grammar, a deterministic parser was developed (Ref 17: chapter two and appendix B). This differs from previous parsers based on transformational grammars which were of the Augmented Transition Network (ATN) type.

The ATN type of parser employs tree search techniques for all syntactically correct solutions. This approach is inherently slower because it extensively uses the time consuming process of backtracking. It is also a less accurate method of parsing English since it produces all correct syntactic solutions instead of the maximally grammatical one(s). Finally, the ATN parser is an unreliable model of the HSPM since it does not fail when the HSPM fails and, therefore is not a desirable syntactic filter for common spoken English.

Deterministic parsing, on the other hand, prohibits backtracking. It works on the realization that "there is enough information in the structure of natural language in general, and in English in particular, to allow left-to-right deterministic parsing of [most]... sentences" (Ref 17:14).

The elimination of backtracking dictates more efficient parsing. A well written deterministic parser is more accurate (in that it psychologically models the Human Speech Parsing Mechanism) and faster since it does not waste processing time (and other resources) constructing parsing paths which will prove to be unsuccessful. Milne explains:

> The assertion of deterministic parsing is that a natural language grammar can be essentially a characterisation of a deterministic machine. However, there are two ways a grammar interpreter using a seemingly deterministic grammar can simulate non-determinism. These are backtracking and pseudo-parallelism.
>
> We can prohibit backtracking by insisting that all grammar substructures are permanent. In a parsing context this means that, if one item is attached to another, this attachment can never be broken... If a word is disambiguated

to a certain part of speech, it can never be changed to a different part of speech... This prevents the grammar interpreter from pursuing a guess that turns out to be incorrect.

It is possible to avoid backtracking, but simulate nondeterminism, by taking all possible paths from a given state simultaneously. This is known as pseudo-parallelism.

This method, however, is still not permissable for a deterministic parser. Using pseudo-parallelism, it is possible to follow each permissable transition simultaneously. If one of the paths fails, the parser does not return to a previous state, but, instead, "throws away" any structure built and then terminates that path. This technique is therefore also dis-allowed. In deterministic parsing, building a constituent and then "throwing it away" is not permitted.

We have two points relating to a deterministic parser. It must neither backtrack nor use pseudo-parallelism. In deterministic parsing, should a transition be made from some state, we are guaranteed that the subsequent state will be on the path to a successful parse, if such a path exists. We shall consider this to be the definition of deterministic parsing (Ref 17:14,15).

Milne has successfully demonstrated that with a two buffer (one word) lookahead buffer, his parser accurately models the HSPM (Ref 17: chapters 3,7,10 & 12). Milne's parser is therefore the ideal choice for use as the English Parser in the SPEREXSYS.

C. Concept of the Solution

On a macro-level, the Spoken English Recognition Expert System (SPEREXSYS) has data flows in accordance with the following diagram:

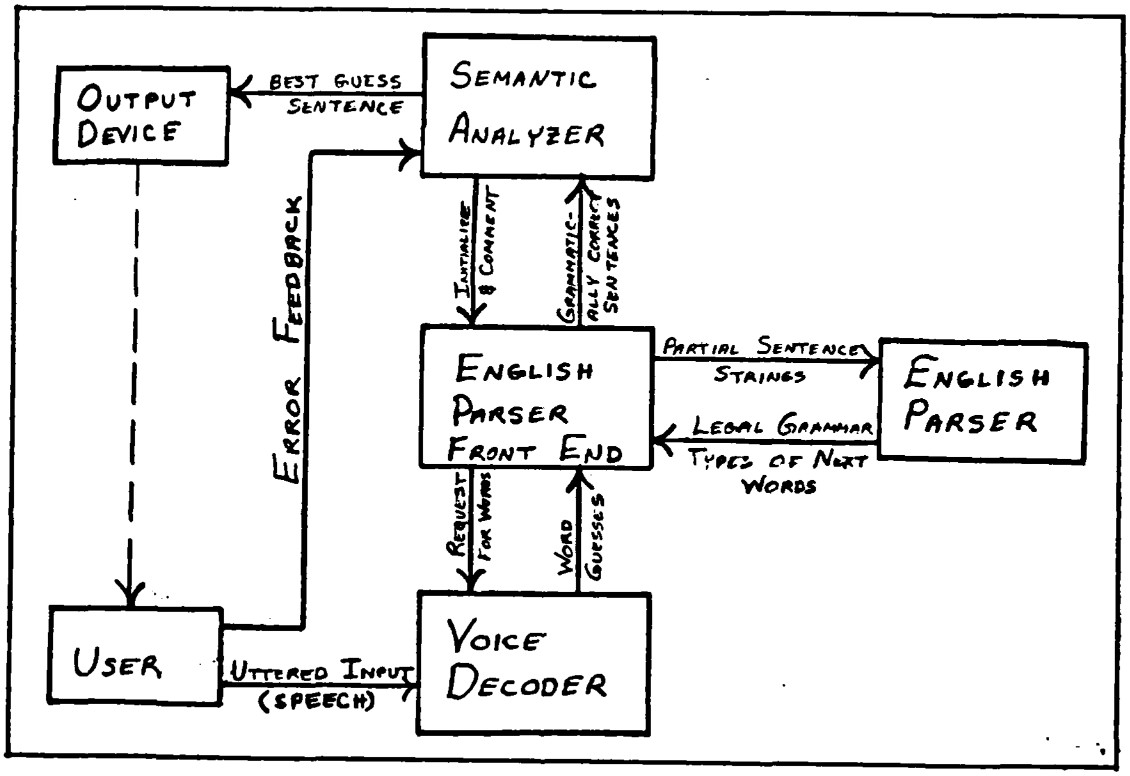


Figure 2.2. Hierarchical Design of the SPEREXSYS

This conceptual model has been chosen because it closely resembles the Levinson model of the HSRS (refer to Figure 1.1). It is important to note that this configuration allows both the English parser (syntactic analysis level) and the semantic analysis levels to review and comment on the likelihood of words (modify the word probabilities) as they are output from the voice decoder. This allows some of the key conceptual features of the Hearsay II blackboard system to be incorpoated into the SPEREXSYS (Ref 7:213-253). Specifically, it allows all levels of syntactic and semantic analysis to cooperate directly and simultaneously to help resolve word recognition ambiguities. This configuration also allows for the modular development and integration of each of the separate functions of the SPEREXSYS as recommended by Montgomery (Ref 18:113).

Having established the conceptual configuration which will govern the development of the SPEREXSYS, it is now appropriate to consider the issues which constrained its design.

Sufficient background knowledge has been reviewed in this chapter so that the required function of the SPEREXSYS can be more adequately defined than was presented in the first chapter.

The SPEREXSYS must apply the constraints of English syntax as defined by the Milne English parser to word guesses of questionable reliability which are made by the voice decoder as it examines the input utterances. Optimally, these syntactical constraints should include the application of both deterministic parsing and the one word lookahead constraints which produce strings that are psychologically correct in that they approximate the strings that would be constructed under similar conditions by the Human Speech Recognition System. In addition, the

SPEREXSYS should be designed in such a way that the upper semantic analysis levels can monitor and influence the word choice decisions when appropriate.

The use of the words "questionable reliability" in the above required function statement refer to the fact that, because of both the inaccuracy of the voice decoder and the insufficiency of information in the acoustics of the speech, the voice decoder may make wrong guesses as to which word was most likely spoken in any given word time frame.

In order to keep the SPEREXSYS from becoming so computationally bound that a real time solution is no longer potentially feasible, it is imperative that sentences be constructed in a deterministic manner. The following discussion illustrates this necessity.

Let us suppose that our voice decoder is so accurate that it will place the proper word (the word actually intended by the speaker/user) within the top three most likely choices 99% of the time. (This is currently beyond the accuracy of state-of-the-art acoustic processors with any reasonable vocabulary size). Let us further suppose that the sentence being spoken is isolated speech (which is easier to recognize than common connected speech) and that a 14 word sentence is being spoken. For example, let the sentence be: "The boy gave the gift to the girl wearing the long green plaid dress." Let us further suppose that the vocabulary size is large enough to provide words which are fairly close in sound to each of the intended 14 words in the sentence. The following matrix illustrates the problem as it has been developed to this point:

probability of likelihood	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
Highest	The	toy	gave	the	sift	to	the
2nd High.	A	boy	cave	a	rift	two	a
3rd High.	They	ploy	save	they	gift	too	they

Word 8	Word 9	Word 10	Word 11	Word 12	Word 13	Word 14
grill	wearing	the	long	green	plaid	rest.
girl	sparing	a	wrong	grease	played	wrist.
curl	bearing	that	gong	clean	glad	dress.

It can be seen that several sentence strings can be constructed from the voice decoder's list of top three choices. Each of these could then be sent to the English parser to determine which of these sentences were grammatically acceptable. Those which are acceptable to the English parser would then be forwarded to the semantic analysis levels for further disambiguation. The following analysis shows that this approach is not feasible.

One sentence which could be constructed for shipment to the English parser is : "The toy gave the sift to the grill wearing the long green plaid rest." Another is: "A toy gave the sift to the grill wearing the long green plaid rest." A third would be: "They toy gave the sift to the grill wearing the long green plaid rest." It can be seen that with three different word choices at each of the 14 different word places in the sentence, that the number of sentences which would have to be constructed and sent to the English parser would be:

$$3^{14} = 4,782,969.$$

Currently, it takes the English parser about a half a second to analyze a sentence. This time could be reduced by a couple of orders of magnitude if the code was optimized and run on a much faster (more expensive) machine. With the best that money can buy, one might reasonably expect to approach one millisecond to process a sentence through the English parser. If all other processing and communcation time in the SPEREXSYS is ignored, it can be seen that under optimal conditions, this 14 word sentence would take: (1 msec/sentence) x (4,782,969 sentences) = 4,782,969 msec. This is equal to one hour and 20 minutes to process a single sentence with a marvelously accurate acoustic processor. Quite obviously, this is an unpractical approach to solving the problem. At this point, the reader might argue that if the analysis were to be completely done in parallel (over 4 million large computers each processing one sentence at a time), that real time processing could be accomplished. The expense, of course, is prohibitive. At this point the reader might argue that such parallel processing capability exists in the human brain. It should also be pointed out that the throughput for the human brain is about 50 bits per second. That is about 400,000 times slower than the fastest electronic computer operating under optimal conditions. It can be shown that it would take the human brain at least several minutes to perform only the syntactic analysis of this one sentence. It is quite clear at this point that performing exhaustive tree searches (such as those used in ATN parsers) is not a feasible approach to solving this problem. Similar calculations rule out exhaustive pseudo-parallel processing.

It becomes obvious that the syntactic analysis must be performed in such a manner so that right decisions are consistently made without having to examine all of the high probability options. Hence, a deterministic approach must be used. At this point, the astute reader will realize that an English parser based on a deterministic approach must be used to solve this problem. It is a great boon to AFIT speech recognition research that the only working (properly) deterministic parser in the world is the Milne English Parser.

The design of the SPEREXSYS incorporates all the aforementioned requirements and is explained in the next chapter.

III. Design

The SPEREXSYS was designed using a top-down hierarchical approach. The top level design was done using data flow diagrams. The intermediate level design was done using structure charts. And the low level design was done using an abbreviated form of pseudo-code which led directly to the actual coding of the SPEREXSYS.

Three programming languages -- C, Pascal, and LISP -- were considered as languages in which the SPEREXSYS would be programmed. The C programming language was considered because of its ability to assist in the problems of connecting the three different computers together (the voice decoder, the English parser, and the rest of the SPEREXSYS all run on different computers). Pascal was considered because its highly structured nature was considered to be a valuable asset in both translating the design into code and in the subsequent testing and debugging of functionally isolated modules. LISP was considered because of the facility with which it can manipulate word strings. The decision to choose one of these languages over the others was not made until the low level design stage. During the pseudo-coding of the design embodied in the structure charts, it became apparent that the list processing capability of LISP was the most important consideration in the timely design of the SPEREXSYS. At this point Pascal was discarded as an option.

LISP was chosen as the primary language in which the SPEREXSYS would be programmed and C was reserved until the end as an option in which the I/O handlers between the computers could be written if LISP proved inadequate for the task. As it turned out, interfacing the computers was an even more difficult task than originally expected. In the end, all I/O and interface requirements were able to be handled by Franz Lisp (the version of LISP which was used for this program), but the decision to discard C as an option was not made until final success was achieved in satisfactorily interfacing the computers. The details of how this was done are covered more completely in chapter four and appendix C.

A. Top Level Design

The top level design specifies the major functional modules and describes the primary data flows needed between the modules.

In the data flow diagram of the entire speech system's interrelation (figure 3.1), it can be seen that the SPEREXSYS interfaces with the voice decoder, an output device, and an input device. The speaker/user speaks into the system input microphone. This uttered input is divided into eight millisecond time slices and analyzed (as previously outlined in chapter two). Eight milliseconds is therefore defined as the basic unit of time. Time zero is

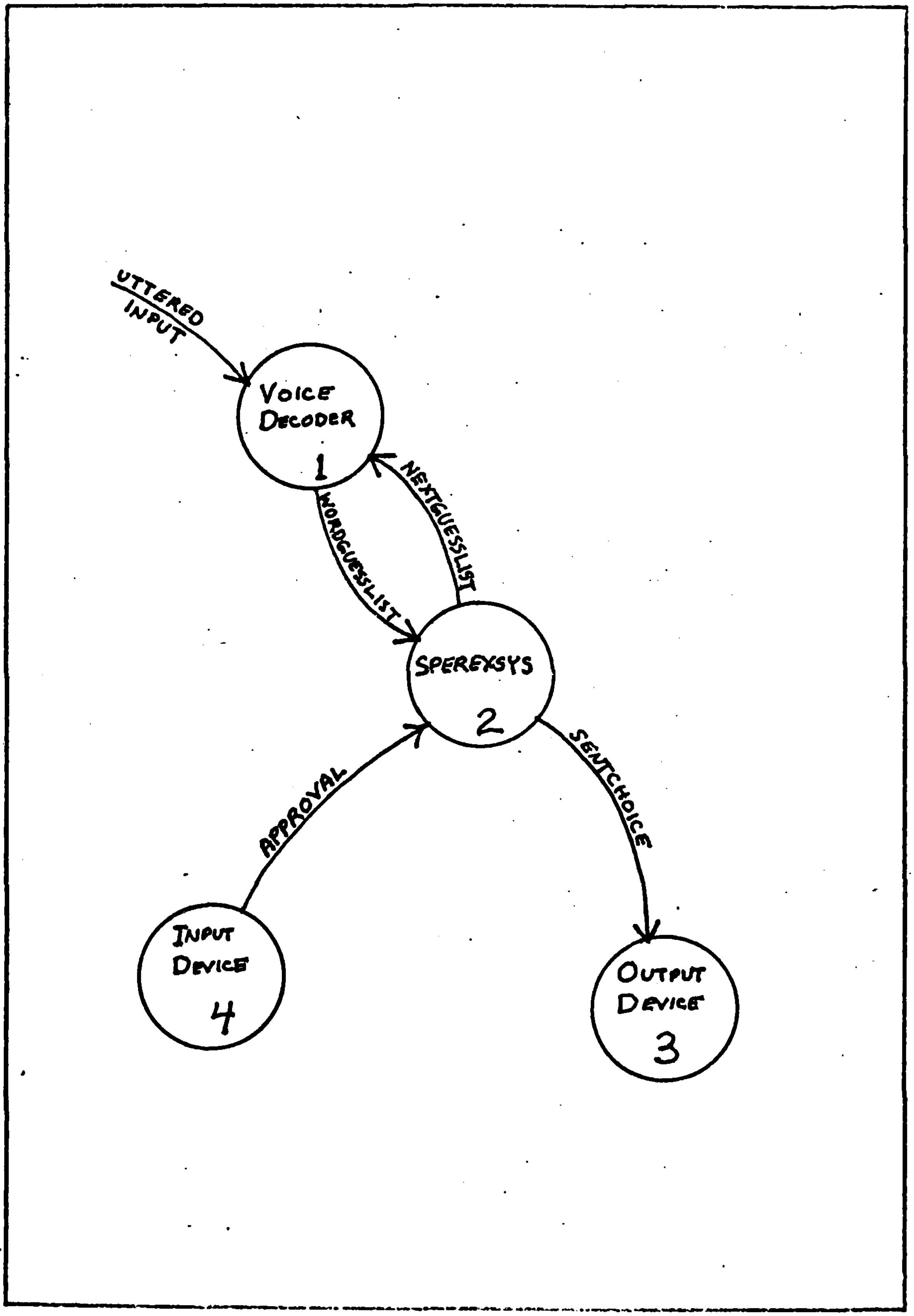


Figure 3.1. Major Components

defined to be the beginning of the first eight millisecond time slice of the utterred input. Time one is the beginning of the second eight millisecond time slice, and so forth.

The SPEREXSYS solicits input to itself from the voice decoder by sending to the voice decoder a list of next word-guess-requests (nextguesslist). Each word-guess-request contains a string number, a time, and a set of grammatical types. The string number is attached to the voice decoder's output (wordguess) for the word-guess-request and is the SPEREXSYS identification tag for that particular sentence string (which, of course, is still under construction). The time is the approximate location of the beginning of the next word in the input utterance. Specifically, it is the exact time at which the previous word terminated. For the first word of the uttered input, this time is zero. (Note - because of the overlap of terminal and next- word-initial phonemes in connected speech, the time parameter passed in the next-guess-request will sometimes mark a point which occurs after the beginning of the next word for which the voice decoder will search). The set of grammatical types specifies the grammatical type constraints which the English parser has placed on the next word to be guessed for that string. These grammatical constraints serve to reduce the effective vocabulary size which must be searched, and hence, improve the reliability of the voice decoder.

In response to each next-guess-request (in nextguesslist), the voice decoder prepares a list of words (wordguess) which fit both the time constraints and the grammatical constraints and which are close enough matches to the input utterance to be likely candidates for the next word in the sentence string. The likelihood of each word (the closeness of match) is annotated by the voice decoder by an assignment of a probability of correctness to each word. The time of each initial phoneme (referred to as tim1 later) and of each terminal phoneme (referred to as tim2 later) are also sent to the SPEREXSYS for all of the words.

This interchange of next-guess-requests and word guesses between the SPEREXSYS and the voice decoder continues iteratively until the SPEREXSYS is satisfied that it has constructed the user intended sentence.

At this point, the SPEREXSYS verifies its results by sending the decoded sentence to the output device (in this case a CRT) and awaits user approval or disapproval of its choice. If the user approves, then the decoding of the next sentence begins where the approved sentence terminates. If the user disapproves of the sentence, then the SPEREXSYS sends the user the next most likely sentence. This continues until the SPEREXSYS finds the right sentence or gives up and asks the user to repeat the sentence with particular care given to the consistent pronunciation of the words which the SPEREXSYS improperly identified.

The data flow diagram which describes the top level design of the SPEREXSYS is drawn in figure 3.2. The data

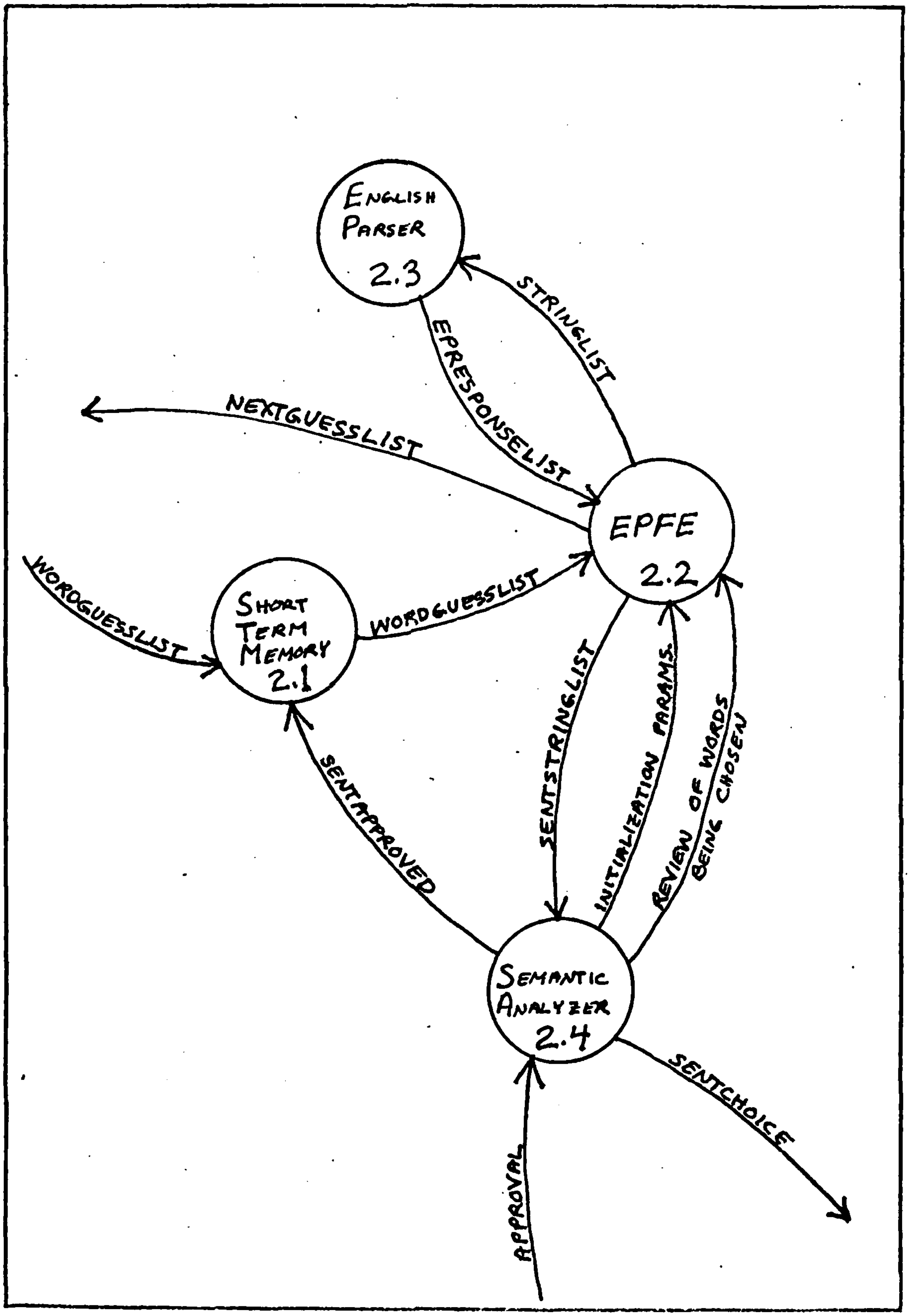


Figure 3.2. SPEREXSYS

flow, as has already been explained, begins with the English Parser Front End (EPFE) issuing a request to the voice decoder for next word guesses (nextguess) for each of the active word strings (an active string is a candidate sentence under construction). The voice decoder responds with a list of possible next words for each active string.

Each of these possible next words is filtered through the short term memory. The short term memory is functionally similar to the psychologically apparent phenomenon of the short term memory in the HSRS. The short term memory in the HSRS favors the interpretation of words which have recently been spoken (in the current conversation) if an ambiguity exists between a recently spoken word and a word which approximately sounds the same. An example which illustrates this phenomenon is presented in appendix E.

If a word which has been recently spoken is one of the words identified by the voice decoder as a possible next word, then the probability of likelihood (assigned by the voice decoder) for that word is increased in the short term memory. The short term memory is updated with the list of all words in a sentence as soon as that sentence receives approval from the user. The short term memory is empty if the sentence being recognized is the first sentence in a conversation; and therefore, for the first sentence, no word probability modifications occur in the short term memory.

The short-term-memory-modified-word-guess-list then is input to the EPFE. In the EPFE (see figure 3.3) the word probabilities are again modified to model a second phenomenon of the HSRS -- this is the phenomenon of longer words having preference over shorter words. The formulation and experimental verification of this hypothesized phenomenon is described in appendix F. The top few words are chosen for each string and the others are discarded.

This abbreviated list of highest probability word guesses is now sent to the start-new-strings module (see figure 3.3) where a new string is started for each word in wordguesslist. Each of these new strings consists of the ancestor string augmented with the new word.

This list of new strings is now sent to the kill-low-prob-strings (see figure 3.3) module which calculates a likelihood of correctness probability for each of the new strings and only keeps the top few most likely ones.

These most likely strings are now sent, one at a time, to the English parser. The English parser analyses each string and provides a list of possible next word grammar types (epresponselist -- see figures 3.2 & 3.3). This list of possible next word grammar types for each string is examined for complete sentences by the formulate-nextguesslist-from-epresponse module. If complete sentences are found, they are reserved for later transmission to the semantic analyzer (sentstringlist -- see figures 3.2,3.3 &

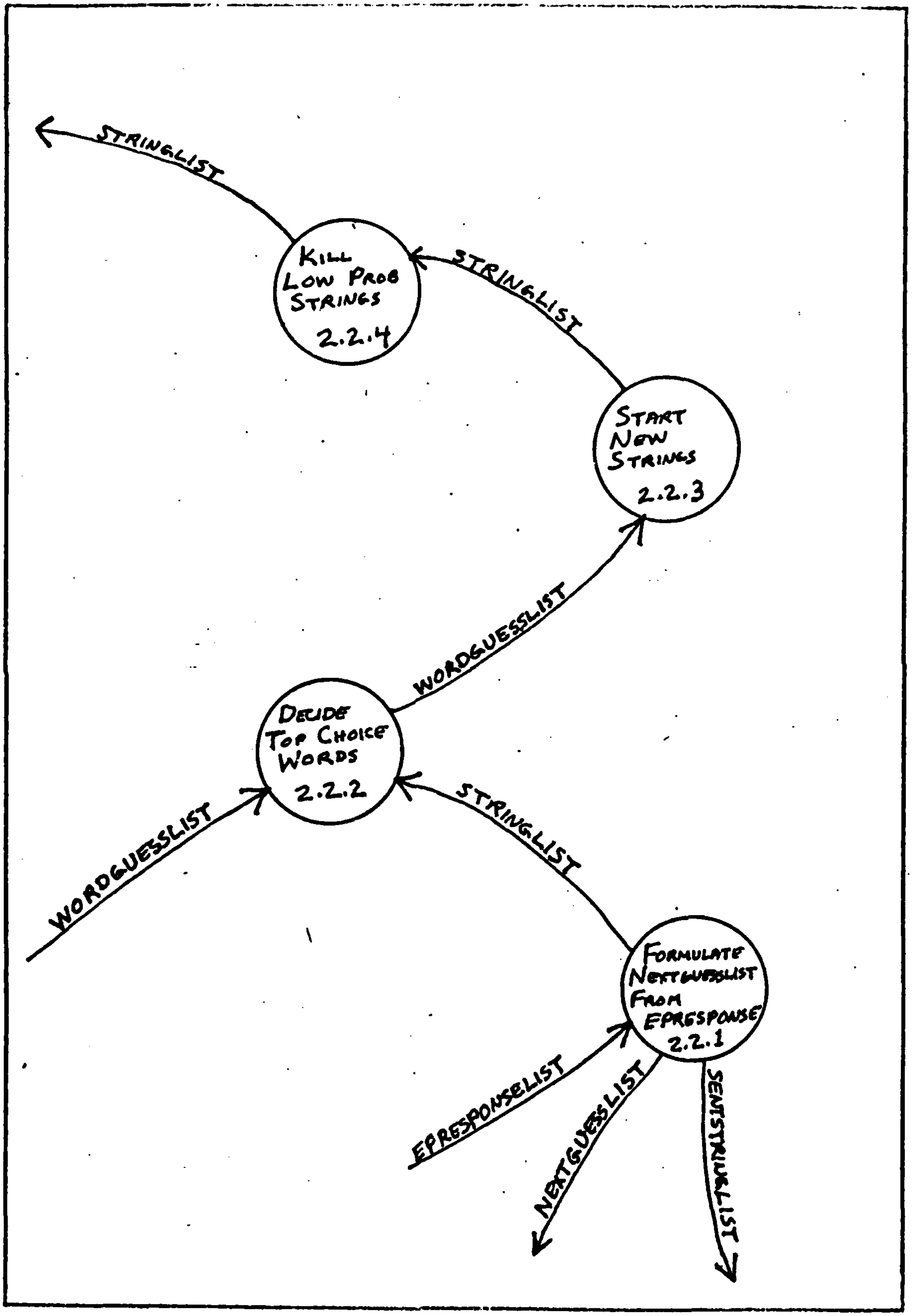


Figure 3.3. English Parser Front End (EPFE)

3.4).

The strings in stringlist are started through the entire cycle all over again by sending the list of possible next word grammar types (in the manner which has been previously explained) to the voice decoder (nextguesslist).

In spoken English, sentences are almost always separated by a brief period of silence. The voice decoder also looks for these periods of silence. When it finds one, it passes that information to the English Parser Front End. When one of these periods of silence (called an FPUNCT -- final punctuation, same as a sentential pause) coincides with a point at which the English Parser has determined that a string could legally be terminated as a complete sentence, this condition is noted in the list of possible sentences (sentstringlist) which the EPFE is storing for later transmission to the Semantic Analyzer.

This continues until either the likelihood of all strings under construction (calculated in the kill-low-prob-strings module in figure 3.3) falls below a user set acceptable threshold, or the voice decoder sends only FPUNCTS (meaning that no further words exist in the input utterance). (Note - the user set acceptable threshold for the likelihood of string correctness is dynamically modified by the semantic analyzer during the operation of the SPEREXSYS to optimize speed and correctness).

The data flow design of the semantic analzyer is shown in figure 3.4 . It is important to remember that the design

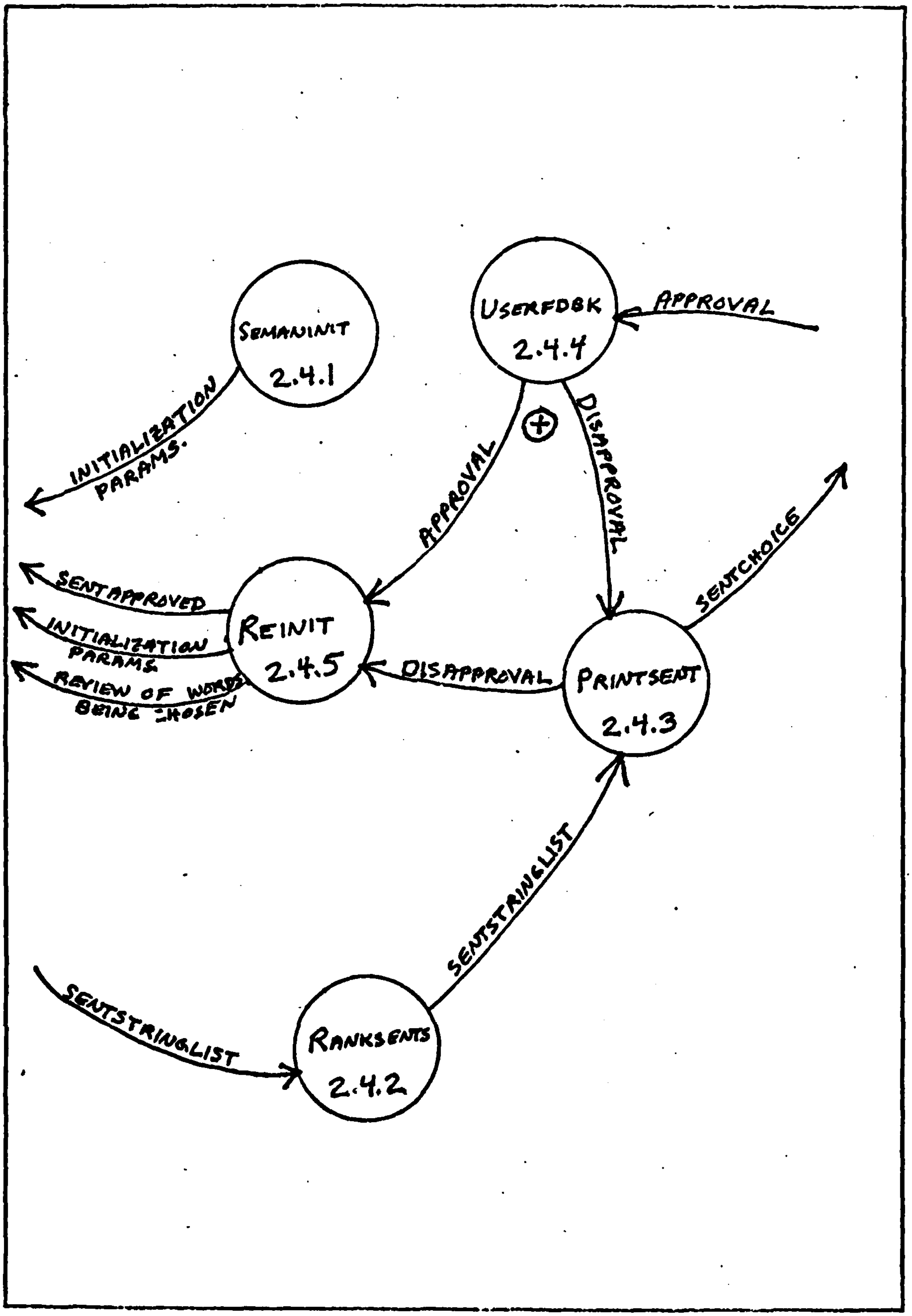


Figure 3.4. Semantic Analyzer

of a psychologically accurate semantic analyzer is beyond the scope of this thesis research. This semantic analyzer functions only as a crude shadow of the functions of the various levels of semantic analysis in the HSRS to the extent that they are understood at all.

At the initialization of the SPEREXSYS, the semantic analyzer requests the user to select the cutoff threshold for the likelihood of correctness of sentence strings, and the number of words which will be accepted to form new strings from each list of possible next words which the voice decoder provides for each string. These and other initialization parameters are passed to the EPFE.

When the EPFE returns the list of candidate sentences to the Semantic Analyzer (sentstringlist), the Semantic Analyzer rank orders each sentence in the list. With a few exceptions, the list will be ordered first to favor the sentences which had coincidental agreement on final punctuation location by both the English parser and the voice decoder, and second to favor longer sentences over shorter ones.

The sentences are printed out to the user one at a time beginning with the most probable. After each sentence is output, the semantic analyzer waits for the user to approve or disapprove its choice. If approved, the REINIT module, reduces the margins of acceptable error, augments the short term memory with the approved sentence, and instructs the EPFE to begin looking for the next sentence in the uttered input at the time the last sentence terminated.

If the sentence is not approved by the user, the next most probable sentence is output. This continues until either the user approves a sentence or sentstringlist is exhausted. When sentstringlist is exhausted and the user still has not approved a sentence, this information is passed to the REINIT module.

At this point, the REINIT module increases the margins of acceptable error, asks the user to repeat his sentence paying particular attention to the pronunciation of the words which the SPEREXSYS failed to properly interpret, and instructs the EPFE to try again.

B. Intermediate Level Design

Structure charts were used to develop the intermediate level design. The structure charts of the entire system were initially drawn only to clarify the modular design. This facilitated the coding of the high level design. Once coding began, these structure charts were extensively modified and expanded. This was due in large part to the recursive nature of LISP. The structure charts presented here are not the originals. The structure charts presented here are those which accurately reflect the completed design.

The entire SPEREXSYS system has been divided into eight major-function diagrams. The structural and functional description of each of these eight diagrams, along with the presentation of the rationale for the key design decisions, is the purpose of this section.

The symbology and conventions used here differ from standard structure chart practices in three major ways. All three of these are because of the nature of LISP.

The first of these differences is that global variables are not shown being passed between modules. It is normally considered poor programming practice to have and extensively use global variables (as this contributes to poor coupling and cohesion). However, in order to take full advantage of the recursive nature of LISP, global variables were used extensively in this design. (No significant problems were encountered during the debugging of the system which could be attributed to the extensive use of global variables!). Only local variables are shown being passed between modules.

The second major difference is that diamonds are not used to indicate decisions. Decisions as to module selection occur for almost every module. (This can be easily seen by examining the number of modules which begin with "cond" statements). Again, this is due to the recursive nature of LISP. It was thought that the use of diamonds would unnecessarily clutter the diagrams making them more difficult to read.

For similar reasons, iterative arrows were not used -- which is the third and final major difference between the standard conventions and those used here.

Figure 3.5 displays the structure of the top levels of the SPEREXSYS. The SPEREXSYS driver (module 0) first calls the SPXSINIT (module 1) which introduces the user to the SPEREXSYS and initializes the system. The driver then calls SEMANALYZER (module 2). SEMANALYZER never returns control to the driver. It is psychologically accurate to do so since the semantic analysis levels represent the highest levels of control in the HSRS. The short term memory module (module 3) is never called by the system driver. It is intended that its function be completely parallel to the rest of the system as it constantly adjusts the probabilities of words at the word selection level in the

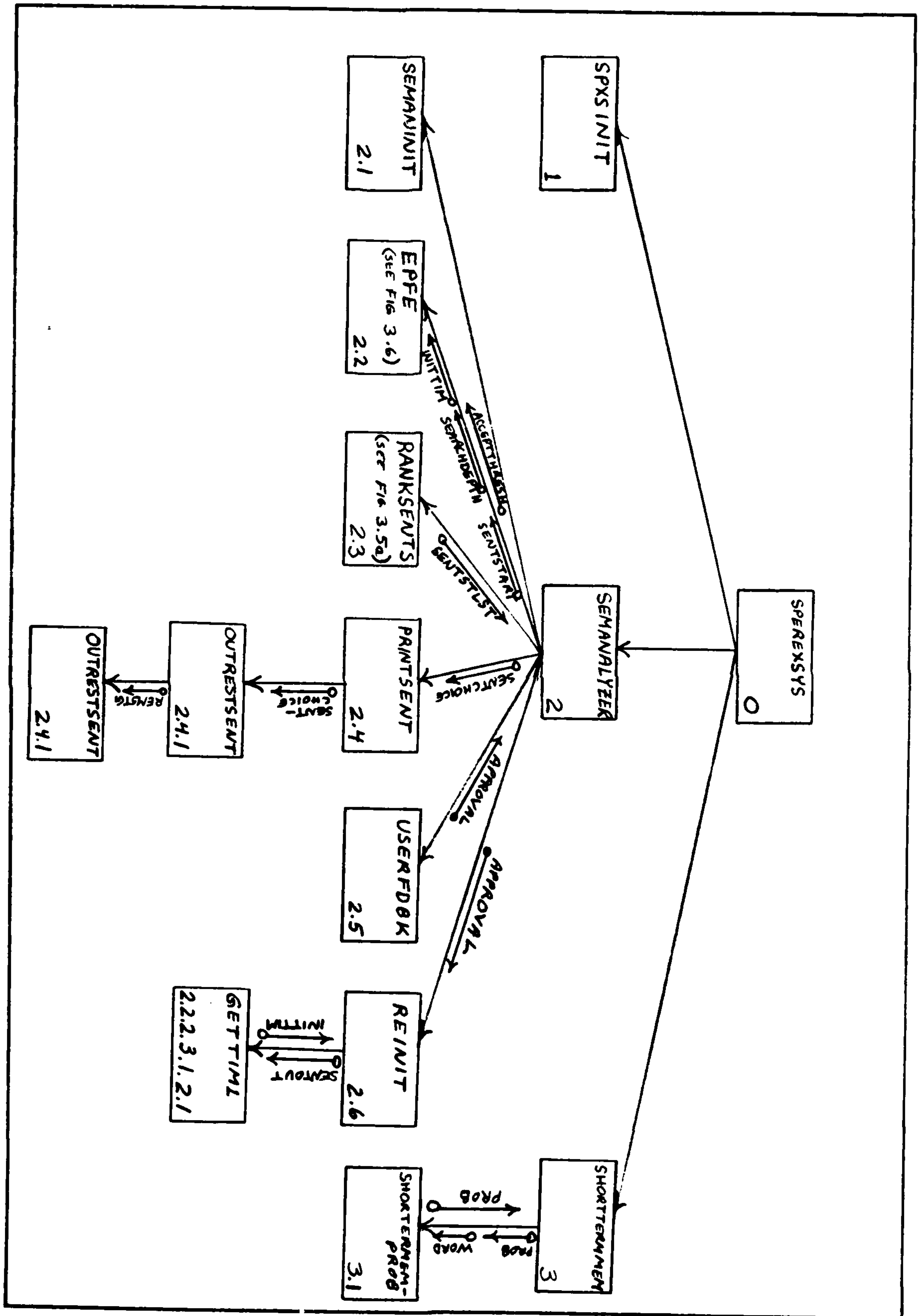


Figure 3.5. Structure chart of SPEREXSYS

EPFE (modules 2.2.*). It is updated upon the positive approval of the user of each decoded sentence. In this system, this is the earliest that updating occurs (as opposed to updating as words are decided within a sentence prior to receiving the complete approval of the sentence). Because of the lack of psychological data, this decision was made to favor the most conservative approach; hence, the short term memory is updated only when the system has been assured by the user that a sentence has been properly interpreted.

It can be easily shown that the short term memory in the HSRS does function to increase the likelihood of selecting a word which has recently been spoken (see appendix E). No data are available on how much the likelihood is increased. Because of this lack of data, it was decided (based on intuition) to increase the voice decoder assigned word probability one-third closer to 1. No claim is made that this is accurate. This probability increase can be easily changed as the equation for it was set aside in a separate module (module 3.1).

The SEMANALYZER module first calls SEMANINIT (module 2.1) which asks the user to set the initial parameters searchdepth and acceptthresh. searchdepth is the number of words deep (the most probable word is at the top) the voice decoder will have to go in order to guarantee (to some user desired degree of reliability) that the correct word will be found. The better the voice decoder is, the smaller searchdepth will need to be. acceptthresh is the acceptance threshold of the average probability of the last three words in a string. If the average probability of the last three words in a string ever drops below acceptthresh, the string is discarded. The use of these terms will be explained more fully later.

Once the initialization parameters have been set, EPFE (module 2.2) is called. The EPFE will be explained more fully later as it is the subject of the next seven structure charts. The EPFE returns control to the SEMANALYZER when it has arrived at a list of candidate sentences.

The RANKSENTS module (module 2.3, Figure 3.5a) is then called. RANKSENTS orders each of the sentences, which EPFE returned, in decreasing order of the probability of likelihood. Two factors are considered when rank ordering these sentences. The first and most important factor is whether or not a sentential pause occured in the uttered input at the same point that the English parser determined that the string was a complete sentence (as previously discussed). Since this is almost a nonvariant phenomenon of human speech (it occurs in all human languages), it is given an overriding emphsis by adding the value of 1 to all string probabilities in which it occurs (this is done in MODFPUNCTS -- module 2.3.1.1). The probabilities for all the words in each string are cummulatively added, along with 100 if a sentential pause occurs, to become the

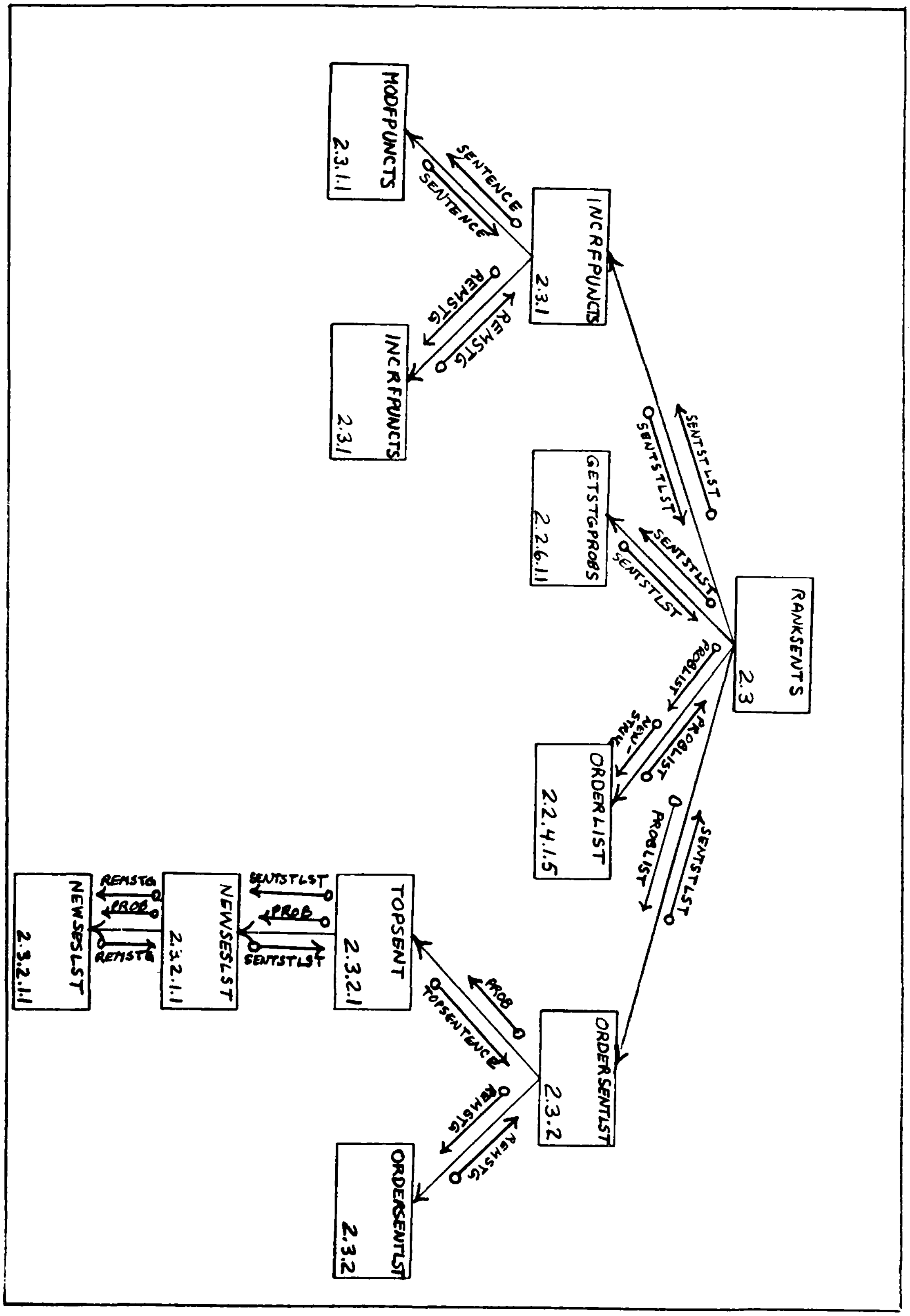


Figure 3.5a. Structure chart of RANKSENTS

sentence probability for each sentence. (Note- This sentential pause, on rare occasion, does not occur at the end of a sentence. Otherwise, sentences without it could be discarded.). The cumulation of all word probabilities in a sentence is the mechanism for favoring longer sentences over shorter ones. The need to do this is illustruated by the sentence:

The boat has nine oars on it.

The following strings are all complete sentences and would be identified as such by the EPFE:

The boat has.

The boat has nine.

The boat has nine oars.

The boat has nine oars on.

The boat has nine oars on it.

The only way to preclude the premature termination of a string is to favor longer sentences. A cursory examination of English conversations reveals that this accurately models the HSRS. If further words continue to make sense as part of a previously completed sentence, then the sentence is continued by augmenting those words.

The list of sentences, along with each sentence's newly calculated likelihood of probability (done in GETSTGPROBS), is sent to ORDERSENTLST (module 2.3.2) along with the rank ordered list of all of the sentence's probabilities (ranked in decreasing order in ORDERLIST). ORDERSENTLST sends the highest sentence probability to

TOPSENT (module 2.3.2.1) which returns the first sentence in sentstlst which matches that probability. NEWSESLST removes that sentence from the old sentstlst. This is so that that any sentence with the same probability will not fail to be selected the next time. (The next probability in problist will be the same as the last one in this case). ORDERSENTLST reorders the list of sentences in decreasing order in this manner. Note that if more than one sentence has the same probability, then the one which appears first in the original sentstlst will appear first in the new rank ordered sentstlst. Without semantics, it is not possible to distinguish between them in any way other than some arbitrary selection such as this.

After the sentences have been rank ordered, PRINTSENT (module 2.4) is called. PRINTSENT prints a banner to the user telling him that the top choice sentence follows. OUTRESTSENT (module 2.4.1) then prints the sentence without all the extraneous information such as word probabilities and times, sentence string number, and sentence probability.

The USERFDBK module (module 2.5) is then called which solicits the user's approval or disapproval of the sentence. If the user approves, REINIT (module 2.6) is called. Otherwise, PRINTSENT is called and the next most likely sentence is printed. If the list of sentences is exhausted before the right one has been found, then REINIT is called and passed this information.

REINIT reinitializes parameters for the next call to the EPFE. If the last attempt to interpret the sentence was successful, then searchdepth is decreased by one (unless it is at two) and acceptthresh is increased by .02. Inittim, the variable which tells the EPFE where to begin looking in the input utterance for the next sentence is set to the termination time of the last word (or FPUNCT) in the approved sentence (done by GETTIM1). The approved sentence is then sent to the short term memory.

If the user did not approve a sentence, acceptthresh is arbitrarily decreased by .05 and searchdepth is increased by two. (Note -- These are different values from the success adjustments to keep the system from ping-ponging back and forth between success and failure). Inittim is left as it was for the last sentence and the user is asked to repeat the sentence more carefully.

Control is passed back to the SEMANALYZER module and EPFE is called again.

When the English Parser Front End is called (module 2.2, figure 3.6), the global variables for the EPFE are initialized by calling GLOBAL (module 2.2.1). GLOBAL also loads two dictionaries -- VOC.DICT and DICT.SPXS. VOC.DICT defines the lists of legal and illegal features (grammar types). DICT.SPXS defines the words in the vocabulary, which are common to all parts of the system, by feature types.

FORMNXGS (module 2.2.2) creates the list of next guess

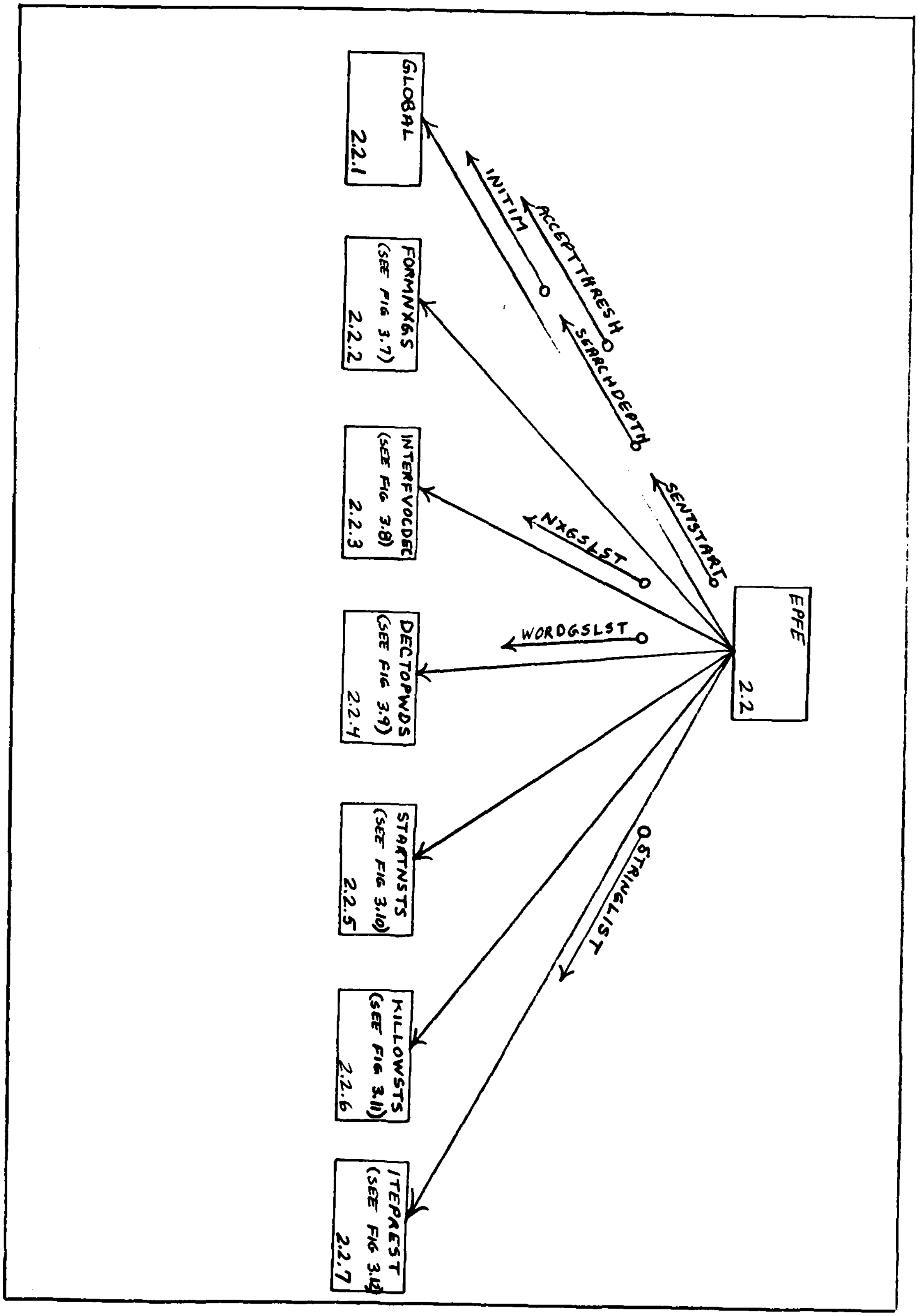


Figure 3.6. Structure chart of EPFE

requests which will be sent to the voice decoder. This process will be explained in more detail later.

Next the EPFE calls INTERFVOCDEC (module 2.2.3) which functions as the communications interface between the EPFE and the voice decoder. This module and its sub-modules (to be described later) output the list of next guess requests to the voice decoder and receive and format the voice decoders response (wordguess) back to the EPFE.

DECTOPWDS (module 2.2.4) is called next by the EPFE in order to choose the top most probable words among those which the voice decoder sent to the EPFE. (This process will be described in greater detail later).

These most probable words are each used to determine new strings by concatenating each of them to the end of its ancestor string. This is accomplished by the START-NEW-STRINGS module (STARTNSTS -- module 2.2.5) and will be explained more fully later.

In order to prevent the number of active strings from becoming larger and larger (it would increase geometricaly by the power of the value of searchdepth if not bounded), the KILLOWSTS module (module 2.2.6) is called for the purpose of selecting only the most probable strings. It is this module which is responsible for accurately reflecting, in the entire system, the deterministic nature of the English Parser. This process will be elaborted on later.

To complete this cycle, ITEPREST (stands for: Iteratively Sends English Parser Response Strings -- module

2.2.7) sends the active strings to the English Parser, one at a time, and forms a list, by string number, of the legal grammar types for the next word in each string. This list is then sent to FORMNXGS and the entire process begins all over again. The details of how this happens, as well as the explanation of how this cycle terminates itself, will be discussed later.

The FORMNXGS module and its sub-modules (figure 3.7) perform the function of interpreting the English Parser's response and translating it into a form which will be understood by and useful to the voice decoder. If the English Parser has identified any strings which cannot be extended to form a grammatically correct sentence, it will not identify any types for the next word. When this occurs, the English Parser's response for that string is said to be nil.

The KILNILSTS modules (modules 2.2.2.1.*) are responsible for eliminating the strings from the active string list when their corresponding English Parser response (refered to in these modules as next2) is nil. This is done by calling LOOKATNEXT2 (module 2.2.2.1.1) for each next2. When LOOKATNEXT2 identifies that next2 is nil, it calls the ELIM module (module 2.2.2.1.1.1) to have the corresponding string eliminated from the active string list and also to eliminate that particular next2 from the English Parser response list.

When this has been accomplished for all the strings in

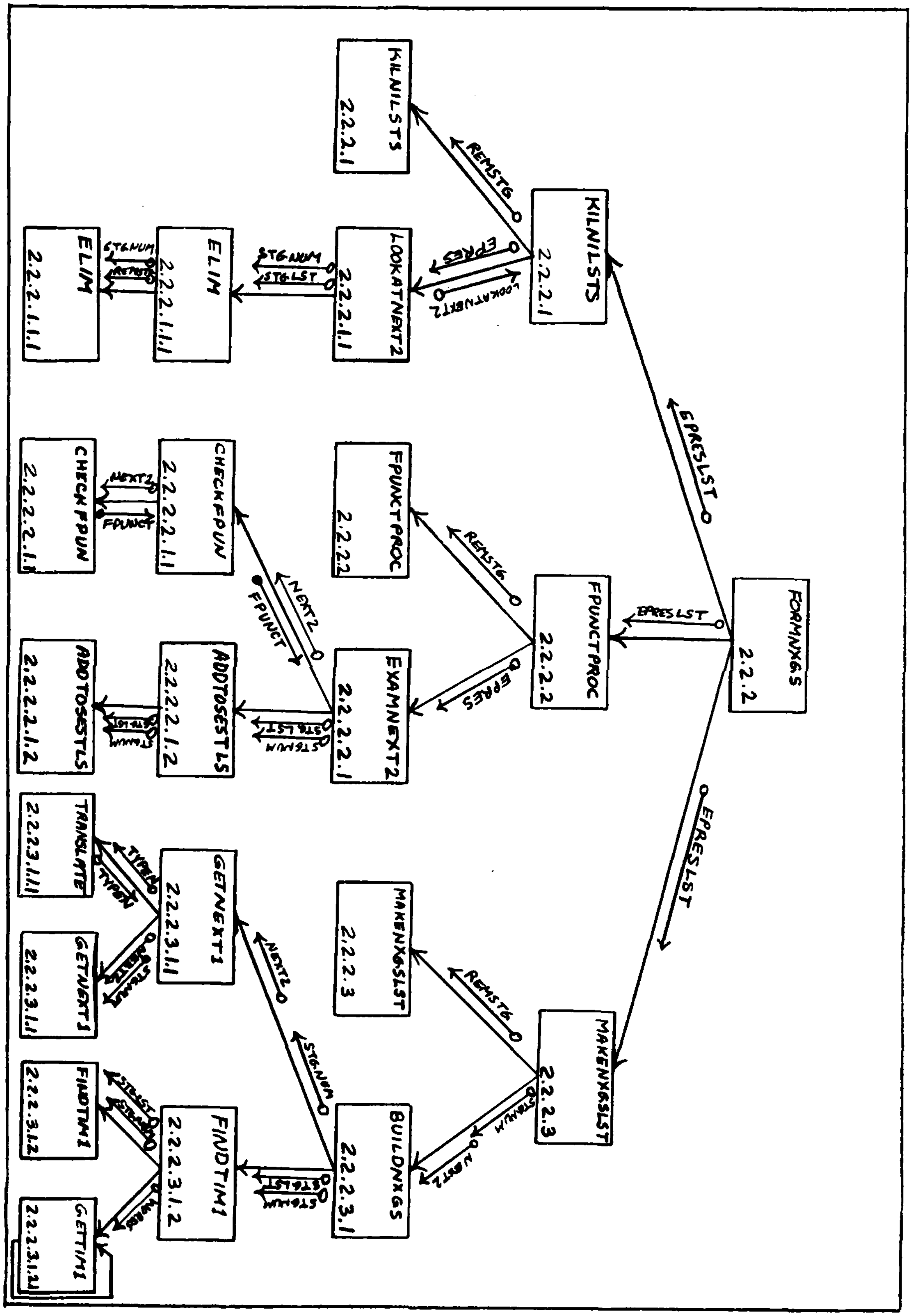


Figure 3.7. Structure chart of FORMNXGS and sub-modules

the English Parser response list (EPRESLST), the FPUNCTPROC module (module 2.2.2.2) is called. FPUNCTPROC calls EXAMNEXT2 (module 2.2.2.2.1) for each next2. Here each next2 is examined to see if any of the legal next word grammar types is an FPUNCT (final punctuation). When it finds such an occurence (found in the CHECKFPUN module), the ADDTOSESTLS module (module 2.2.2.2.1.2) is called which adds that completed sentence to the sentence list being built for transmission to the Semantic Analyzer.

After nil strings and FPUNCTS are taken care of, MAKENXGSLST (module 2.2.2.3) is called. This module and its sub-modules are responsible for the final formation of the list of next word guess requests which will be sent to the voice decoder. One at a time, the next2's are sent to BUILDNXGS (module 2.2.2.3.1) where the grammatical types in next2 are translated into the grammatical types which the voice decoder will understand (done in modules 2.2.2.3.1.1 nd 2.2.2.3.1.1.1). This translation of next2 is now referred to as next1. The time in the input utterance which will be used as the approximate starting point for the next word is found by getting the termination time of the last word in the string (done in modules 2.2.2.3.1.2 and 2.2.2.3.1.2.1). The next guess request for that string is then formed by concatenating the string number and the new word starting time to the next1 for that string. This new list constitutes the next guess request for that string and is concatenated to the list of next guess requests.

Now that the EPFE is prepared to output its list of next guess requests to the voice decoder, INTERFVOCDEC is called. INTERFVOCDEC (module 2.2.3, figure 3.8) is responsible for the interface between the EPFE and the voice decoder.

Normaly, the way to proceed at this point would be to simply print the list of next guess requests out the port connected to the computer which the voice decoder is running on. This could be accomplished with a single three word LISP command. For reasons which will be only partially discussed here and more fully discussed in chapter four, this module was built to do considerably more processing than simply outputting the list of next guess requests.

In order to analyze how effectively the English Parser generated legal-next-grammatical-types were reducing the vocabulary size which the voice decoder had to consider, a list of all the vocabulary words (from the entire 200 word vocabulary) was found and printed which met the constraints imposed by the English Parser. When this was done, a message was printed which told the user how many words were in this reduced list. This allowed for continuing analysis of how much the English Parser was improving the reliability of the voice decoder (reference the previous discussion on this subject).

The English Parser's feature list contains complicated expressions of set unions, intersections, and compliments. An example of one of these complicated feature expressions

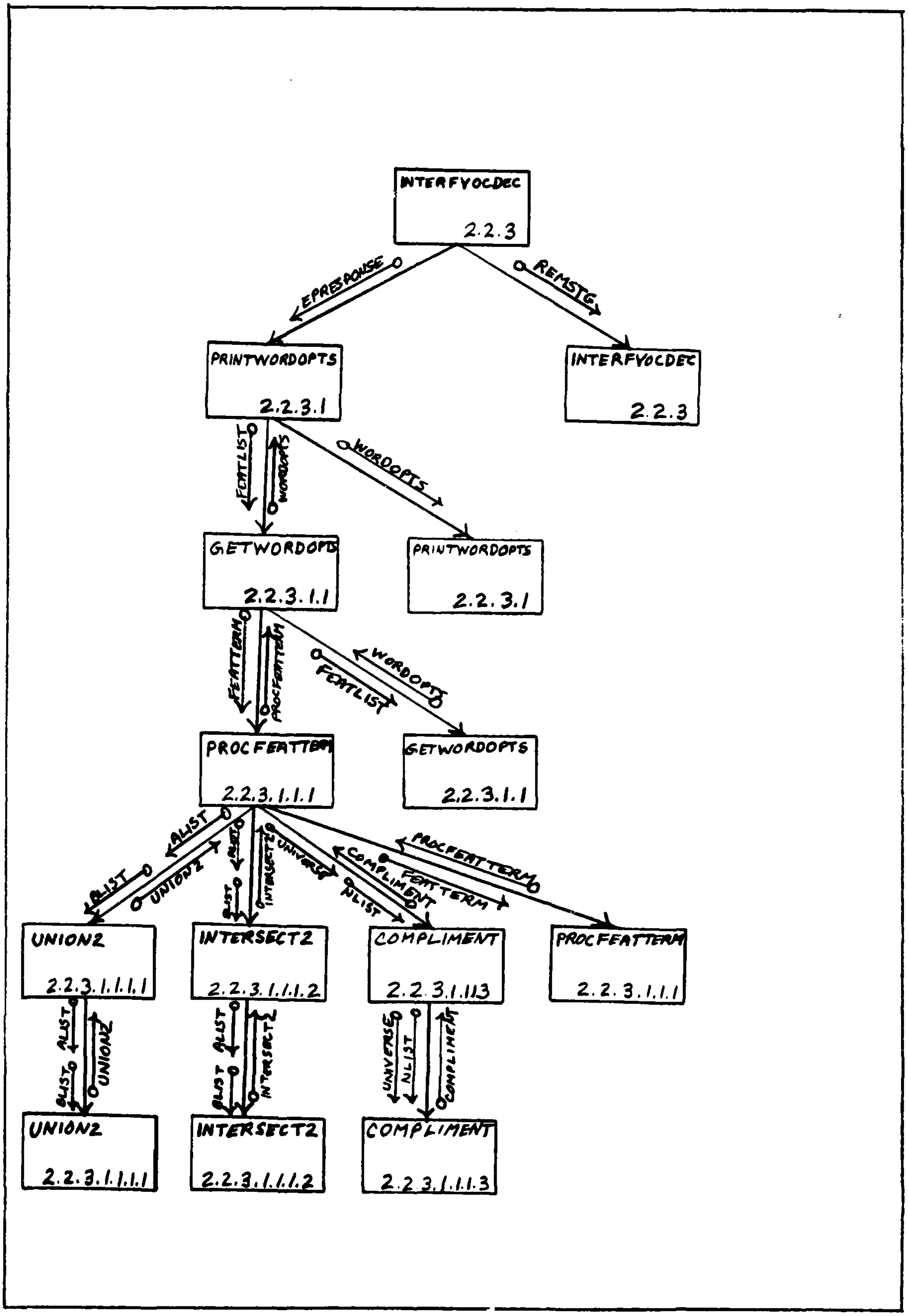


Figure 3.8. Structure chart of INTERFVOCDEC and sub-modules

(which would be part of the epresponse) might be:

[noun or (verb and not (adj or adverb))].

This is to be interpreted as the set of words which are nouns unioned with the set of words which are verbs -- but the verbs cannot be either adjectives or adverbs (unless, of course, they are nouns).

In order to interpret these complicated set expressions and produce a set of words which conform to these constraints, PROCFEATTERM (module 2.2.3.1.1.1) is called which recursively disassembles each feature list and transforms it into a legal set of words. In order for it to accomplish this, union, intersection, and complimenting set functions were written for its use (modules 2.2.3.1.1.1*).

After each epresponse (an element of the list of English Parser responses) is processed and output, the INTERFVOCDEC module waits for the voice decoder's response (wdgs -- word guesses -- reference previous discussion on this subject). As each voice decoder response is received, it is concatenated on to the list of word guesses. This continues until the entire list of next guess requests has been processed.

The user has already informed the SPEREXSYS (at system initialization) as to the maximum depth the voice decoder will have to go to guarantee that the correct word has been recognized. This value was assigned to the variable "searchdepth." Therefore, it makes sense at this point to trim all of the voice decoder's responses for each string to the top "searchdepth" probability words in order to conserve processing resources (most importantly processing time). Instead of just chopping the list off below the third highest probability word, some modification of word probabilities is done at this point which model the psychological processes of the HSRS. As have been already discussed, these are the phenomenons of increasing likelihood of selection for both longer words and recently spoken words.

The primary purpose of the DECTOPWDS module (module 2.2.4, figure 3.9) and its sub-modules is to accomplish the above functions. In addition, if a sentential pause (refer to previous discussion of this subject) has been sent by the voice decoder, it is here that it is detected (done by FINDFPUNCT -- module 2.2.4.1.1) and added to the end of the appropriate sentence in the list of sentences being reserved for later transmission to the semantic analyzer (done in AUGSENSTG -- module 2.2.4.1.2). If no matching sentence is found in that list, then the occurence of a sentential pause is ignored.

After this has been accomplished, the short term memory reviews every word in the list. If it finds any that have been spoken in recent past sentences (since the start of the conversation), it increases their probabilities by moving them one-third closer to 1.0. This equation is arbitrary because of the lack of psychological data which provides an accurate quantification of the probability

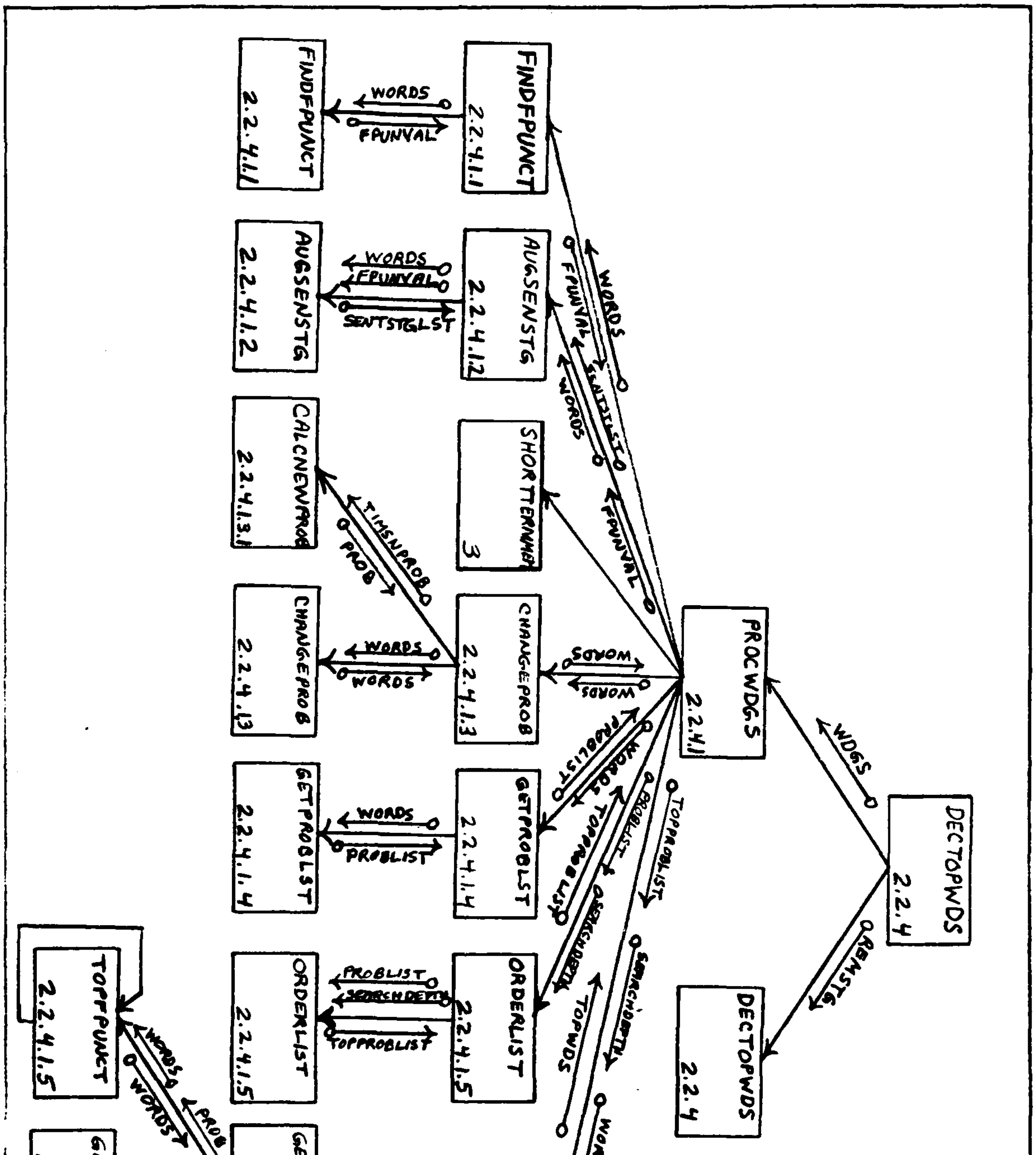


Figure 3.9. Structure chart of DECTOPWDS and sub-modules

increases.

After the short term memory modifies the probabilities of any recently spoken words, the CHANGEPROB module (module 2.2.4.1.3) modifies the probabilities of each word based on the following equation:

$$\text{new prob} = 1/2\ (\text{tim2} - \text{tim1})/(\text{maxwordtim})(1 - \text{old prob}).$$

This equation was isolated from the rest of the CHANGEPROB code by putting it in the CALCNEWPROB module (module 2.2.4.1.3.1) so that it can easily be changed. This was thought to be necessary because it too is an arbitrary specification of the increase of word probabilities because of the lack of psychological data. The above equation modifies word probabilities in only a very minor way, but it is enough to prevent a word boundary from being interpreted prematurely because of a part of the word also being a very close match to the uttered input. For example, it would prevent the word "ambiguous" from being interpreted as the four words "am big you us."

GETPROBLST (module 2.2.4.1.4) now strips off the probabilities of each word for a single voice decoder word guess response and sends them to ORDERLIST (module 2.2.4.1.5) which rank orders them in decreasing order. FINDTOPWDS (module 2.2.4.1.6) then gets the Nth element in the list where N is the value of searchdepth. (Done in TOPFUNCT through GETTOPWDS). This minimum acceptable probability is sent to STRIPTOPN (module 2.2.4.1.6.2) along with the list of words from the word guess voice decoder response. STRIPTOPN discards all words with probabilities less than the minimum acceptable probability.

This procedure continues until each word guess response has been processed.

Now that only the top searchdepth number of words are still active for each string, this reduced list is sent to STARTNSTS (module 2.2.5, figure 3.10) where the new strings are formed.

In the event that this list is empty (which would occur if the voice decoder only sent back FPUNCTS -- signaling that the end of the user's uttered input has been reached), the EPFE will return control to the SEMANALYZER module of the Semantic Analyzer.

STARTNSTS sends the entire stringlist (list of active strings) to NEWSTRINGS (module 2.2.5.1) which sends one string at a time to FINDWDSMATCH (module 2.2.5.1.1). FINDWDSMATCH does two things. First it sends the string number (of the string it is working on) to GETWORDS (module 2.2.5.1.1.1) which returns the top searchdepth words corresponding to that string number, then it sends the string and its new top next words to MAKESTS (module 2.2.5.1). MAKESTS makes new strings, one for each of the top searchdepth next words, by appending the word to the end of the string, giving that string a new unique string number, and concatenating that string to a variable called

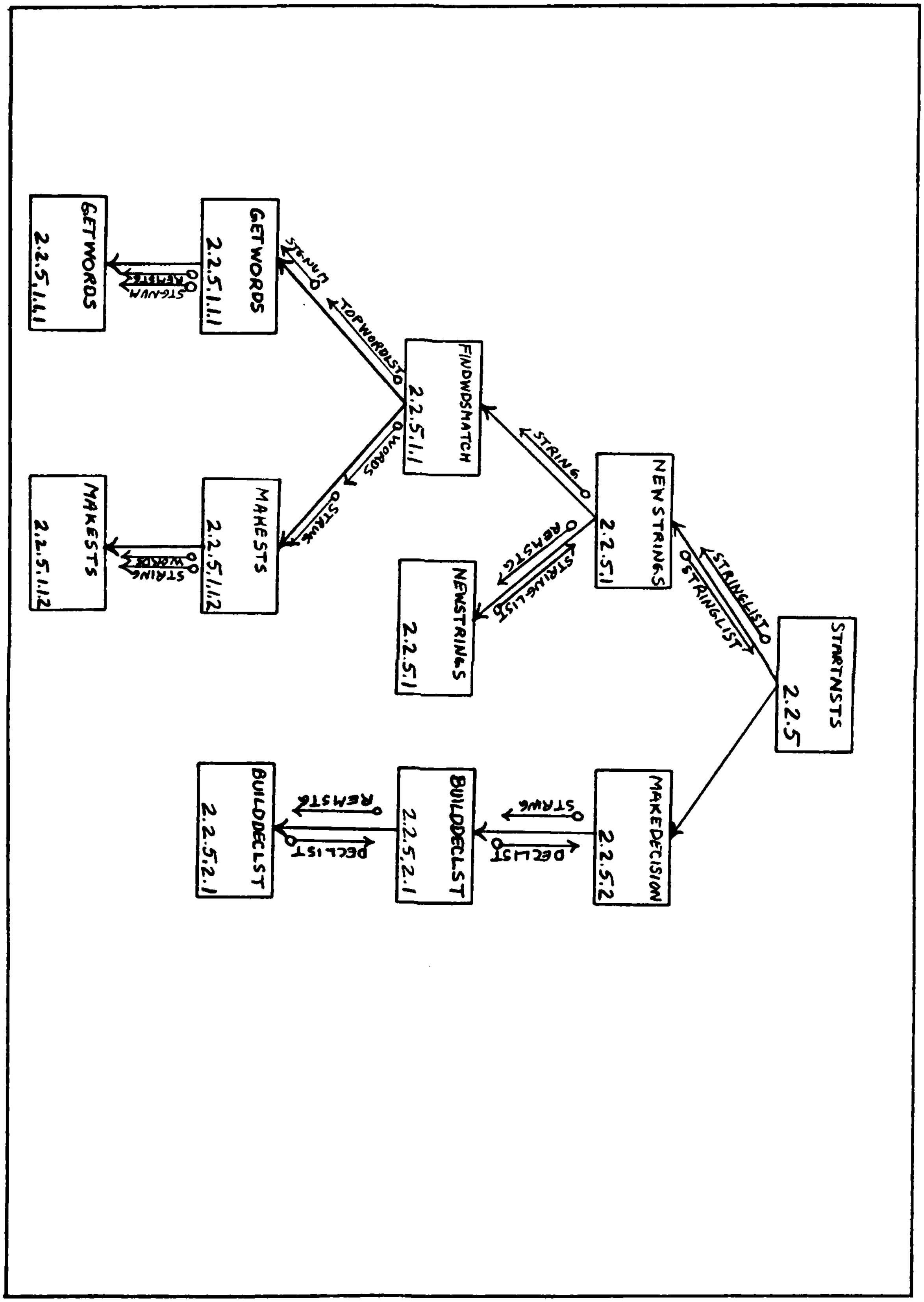


Figure 3.10. Structure chart of STARTNSTS and sub-modules

OPTSTGLST. OPTSTGLST was set to the value of an empty list before entering the STARTNSTS module. Before exiting the STARTNSTS module, STRINGLIST is assigned the value of OPTSTGLST. In this way, old strings (the ancestor strings of the new strings) are all eliminated, and only their high probability children are allowed to continue and compete for survival in the KILLOWSTS module (module 2.2.6). But before they are allowed to continue on to the KILLOWSTS module, they are first sent to MAKEDECISION (module 2.2.5.2) where the user is informed of the list of third words from the end of every string. In maintaining the psychological accuracy of modeling the HSRS which the English Parser provides through the use of its one word lookahead theory. Because this one word lookahead relies on the fact that there is no uncertainty as to the proper identification of each word, and since this word identification and word boundaries are not yet known with only one word lookahead, it has been deemed appropriate to use two word lookahead for the determination of those strings which will continue to survive. Syntactic function will continue to be assigned to guessed words based on a one word lookahead. This compromise is expected to maintain the psychological similarity of the HSRS while allowing for the string to develop further before making a final decision on the proper word in a given word place. It is necessary to make a decision on which of these third words back from the end of the active strings is really the correct word.

It is critical to an adequate understanding of this thesis that the reader fully grasp why a decision is being made now on the third word back from the end of all active strings.

IT IS AT THIS POINT IN THE PROCESS THAT THE SEMANTIC ANALYZER, WHICH WILL MAINTAIN THE UPPER LEVELS OF SEMANTIC ANALYSIS, TO INCLUDE THE SEMANTIC NETWORK DESCRIBED IN CHAPTER 5 AND APPENDIX D, WILL HAVE TO COMMENT ON THE WORD SELECTION IN THE EPFE. Any semantic commenting about words prior to their appearance as the third word from the end of a string (with the exception of the last two words in a complete sentence) is probably premature. The Parser cannot be reasonably confident that it understands the function (grammatically) of a word until it is able to see the next word. The semantic analyzer cannot comment on the reasonableness of a word (based on its meaning) until it understands the function of that word (i.e. -- whether it is supposed to be a noun, or a verb, or an cdjective -- most nouns can function as either of these three). Therefore, it follows that the semantic analyzer will not normally be asked to comment on the likelihood of a word until it is followed by at least one other word. But again, because of word identification ambiguities, a more informed decision can be made using a two word lookahead instead of only the one word lookahead which Milne's theory dictates.

Figure 3.11 illustrates the structure of the KILLOWSTS

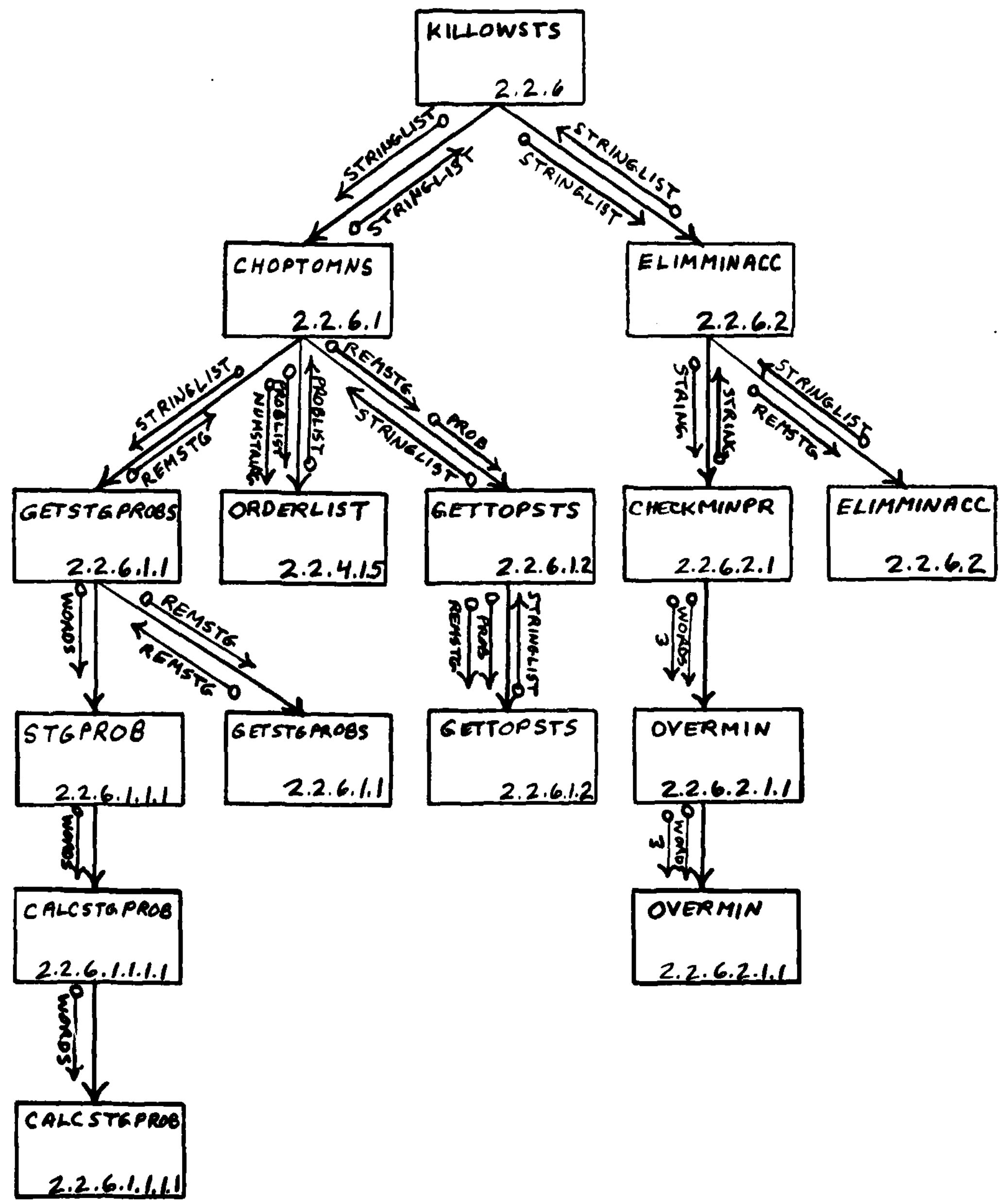


Figure 3.11. Structure chart of KILLOWSTS and sub-modules.

module and its sub-modules. The number of active strings in the system would increase geometrically by the power of the value of searchdepth if they were not selectively deleted. It is the responsibility of these modules to selectively delete all but the top few most probable strings. In order to do this, probabilities of likelihood must be assigned to all strings.

It was initially decided that only the last three words of each string should be considered in determining a string's probability of likelihood. This was in order to incorporate all of the psychological modeling of the HSRS which Milne demonstrated was attainable if the decisions on word identities were made based only on the next two words in the sentence. In order to do this, it was initially envisioned that the EPFE should make a decision on the third word back from the end of a string (all strings in the system would be identical up to the fourth word back from the end of the string). After further consideration of this proposed constraint, it seemed unreasonable to force a decision on the third word from the end of the sentence only because that was the point at which the HSRS made syntactic decisions. It became evident as this was discussed that since the identity of a word requires semantic (not only syntactic) judgement, and it was known that not all semantic decisions were made on a word when only the next two words were known, a design compromise was made to calculate string probabilities based on all of the words in a string, and then to choose the top n-squared strings as the survivors. In connected speech, this does not force a decision on the third word back from the end of a string, but allows ambiguity of the third word back from the end of the string if the cummulated probabilities of all the words in a string are high enough to compete with the other survivor strings. Note that for separated word speech, where the word boundaries are known, this would still force the decision to make the third word back from the end of all active strings identical. Tree search diagrams were used to prove this conclusion.

Further, it was decided that when the cummulative probabilities of the last three words of any string were less than the minumum acceptable threshold (acceptthresh), that the string would no longer be considered.

The CHOPTOMNS module (module 2.2.6.1) is responsible for eliminating all but the top searchdepth-squared highest probability strings. GETSTGPROBS (module 2.2.6.1.1) calculates string probabilities (done in CALCSTGPROB), and then makes a list of these probabilities (done in STGPROB). This list is sent to ORDERLIST (module 2.2.4.1.5) to be ordered in decreasing order. This ordered list is then sent to GETTOPSTS (module 2.2.6.1.2) which returns only the top searchdepth-squared strings.

These top strings are then sent to ELIMMINACC (module 2.2.6.2) where the probabilities for the last three words are calculated (done in OVERMIN) and compared with acceptthresh (done in CHECKMINPR). Those which do not pass this test, are eliminated from further consideration.

If no strings have survived to this point (are still active) then the EPFE returns the list of accumulated complete sentences to the Semantic Analyzer and control is passed back to SEMANALYZER.

Those strings which do survive, are sent to the English Parser by ITEPREST and its sub-modules (Figure 3.12). ITEPREST pulls one complete string at a time off the list of active strings and sends it to INTERFEP (module 2.2.7.1). INTERFEP functions as a driver for its sub-modules. First, STGPRINT (module 2.2.7.1.1) forms each string into a command which the Parser can understand and outputs that command to both the Parser and the user's terminal. Second, STGPRINT reads the English Parser's response and concatenates it, with the appropriate string number, to the new list of English Parser responses (epreslst).

This new list of English Parser responses is sent to the FORMNXGS module and the entire cycle is started again.

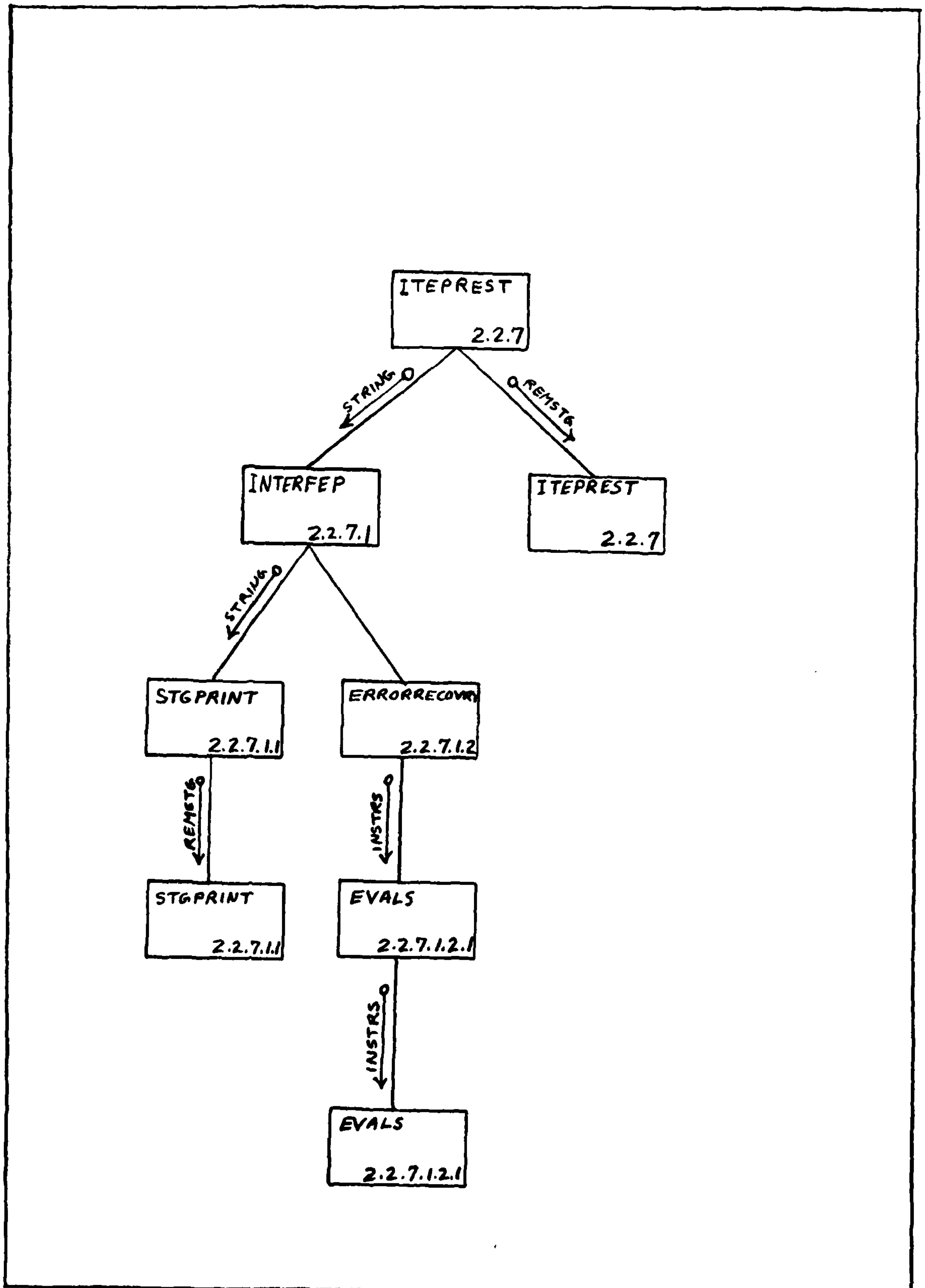


Figure 3.12. Structure chart of ITEPREST and sub-modules

C. Low Level Design

The low level design of the SPEREXSYS was done with pseudo-code. Standard pseudo-code is pascal-like in its structure and terminology. LISP does not look much like PASCAL in either instruction function or in structure. Once LISP was chosen as the language in which the SPEREXSYS was to be written, very unconventional pseudo-code was the result. It was, at best, an informal system for annotating how the functions were to be structured and coded. In the end, it was useful for assisting in the coding of about half the modules. The others, especially the interface and the lowest level recursive modules, were written as their need became apparent. The original pseudo-code was not modified first. This was largely do to the growing realization (LISP was a very new language to this researcher at the outset of this project and only a very shallow understanding of how to properly use it had been achieved) that the structural thinking processes which take full advantage of the recursive power of LISP are not easily described in pascal-like pseudo-code.

For these reasons, it has been decided that the commented listing (appendix A), the data descriptions in the data dictionary (appendix G), and the preceding intermediate level design narrative would be sufficient to describe the low level design of the SPEREXSYS.

IV. Implementation, Testing, and Validation

This chapter is devoted to explaining the specifics of the operation and results of the SPEREXSYS. When reading the section on the implementation of the SPEREXSYS (section A), it will be useful to also reference appendix C which is a short user's manual on how to set up and operate the SPEREXSYS.

Appendix B is an example run on the SPEREXSYS attempting to recognize the sentence: "The peak got snow." It will be useful to reference this appendix in order to better understand the discussion in section B on testing and validation.

A. Implementation

The first and most important item of discussion in the implementation of the SPEREXSYS is that the Voice Decoder was not finished in time for integration into the system. This necessitated the simulation of the Voice Decoder's operation by a semiautomated process under human user control. This did not significantly impact the testing of the SPEREXSYS since the SPEREXSYS was designed to treat the Voice Decoder as a black box with a very limited and strictly defined data transfer between the SPEREXSYS and the Voice Decoder. Any kind of voice decoder (isolated or connected word, small or large vocabulary, any type of feature set extraction, any bit rate, any kind of word recognition scheme) could be used as long as it has the following attributes:

1. It can remember the input utterance.
2. It can determine the acoustic likelihood of match of words in the vocabulary which might be the next word in the input string beginning at or around some specified time in the input utterance.
3. It can identify the start and stop times of every word which it determines to be a likely match.

It was therefore decided to use the acoustic analyzer of the HSRS as the black box voice decoder since nothing else was ready for integration. It should be noted that the HSRS voice decoder which was used purposely made misjudgements as to word identification likelihoods in order to test the flexibility and responsiveness of the SPEREXSYS. These are specifically described in section B of this chapter.

In order to assist the human user in making the appropriate voice decoder decisions, the process of word selection was semi-automated by printing out only those words of the vocabulary which meet the grammatical restrictions which the English Parser placed on the next word to be guessed for each active string. This insured that the human voice decoder would only pick next words which were grammatically acceptable.

The output from the SPEREXSYS to the Voice Decoder was sent to the C.R.T. of the SPEREXSYS user's terminal in order to be able to keep a script (log) of all the data exchanges. For the same reason the input to the SPEREXSYS from the Voice Decoder was input at the keyboard of the SPEREXSYS user's terminal.

The interface between the VAX computer in the AFIT/EN building (on which the EPFE and semantic analyzer ran) and the DEC-10 computer in the Avionics laboratory building (on which the English Parser ran) was a little more complicated and difficult.

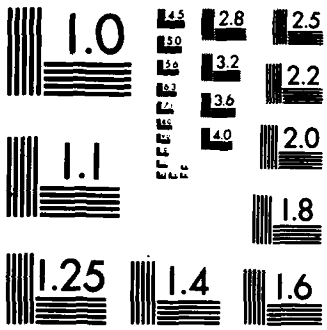
It required the use of at least four terminals and four modems. On occasion, up to seven terminals (with modems) were used. The additional three were helpful in the tasks of line control (between the VAX and the DEC-10) and systems information management. Only the function of the four essential terminals (and their modems) will be described here. Also, a special RS-232 cable was constructed which crossed the wires between pins #2 (RxD) and #3 (TxD) on the connectors for both ends of the cable and connected the #7 (GND) pins of both connectors together.

For the purposes of this description, the four terminals which were used will be referred to as :

TTY11 -- set to 300 baud with telephone modem,

AD-A136 835 A SPOKEN ENGLISH RECOGNITION EXPERT SYSTEM(U) AIR FORCE 2/3
INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF
ENGINEERING R L ROUTH SEP 83 AFIT/GCS/EE/83S-01

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

TTY12 -- set to 300 baud with Gandalf modem,

TTY13 -- set to 9600 baud with Gandalf modem, and

TTY14 -- set to 9600 baud with Gandalf modem.

All four terminals were located in the terminals room (room 125) of the AFIT/EN building (buildling 640). All of the Gandalf modems were connected to the VAX on which the EPFE and Semantic Analyzer ran.

TTY11 was used to call into the DEC-10 computer and initialize the Parser for operation. In addition, it was necessary to turn off the echo of the input in order to keep from sending the input back out the output channel (reference the RS-232 special cable connection below).

TTY13 was brought up and left in the UNIX C-shell. This terminal was used as a dummy input terminal for TTY12.

TTY12 was brought up. Its protocols were switched to the DEC-10 protocols. Its echo was then set to off for the same reason as outlined for TTY11 above. The LISP interpreter was entered and the command:

```
(setq piport (infile '/dev/tty13)) <cr>
```

was issued so that the shell which monitored this terminal's input would not interfere with the input that was coming from the DEC-10. The cables connecting the TTY11 and TTY12 terminals to their respective modems were unplugged from the modem side. The special RS-232 cable was then used to connect the modems for these two terminals together.

Finally, the EPFE and Semantic Analyzer portions of the SPEREXSYS were brought up with the "(load 'spxs)" LISP command on TTY14 which then functioned as the SPEREXSYS user's terminal.

The last thing to be done to complete the preparation of the entire implementation of the system was to send a carriage return to TTY12. (This was done using a LISP command during SPEREXSYS intialization. The carriage return of course did not go to TTY12. It went to the output line of the TTY12 modem which then went into the DEC-10 through the input line of the TTY11 modem). This had the effect of loading a vertical bar into the input buffer for the TTY12 modem to access. This was necessary to properly synchronize the EPFE and English Parser I/O channels because the EPFE is programmed to ignore everything up to the first vertical bar in the TTY12's input buffer (when it talks to the English Parser).

Whenever the EPFE now wants to talk to the English Parser, it uses LISP commands to set its input and output ports to TTY12. When it wants to talk to the SPEREXSYS user's terminal, it resets them to TTY14.

A more detailed set-up procedure is described in the user's manual in appendix C.

B. Testing and Validation

Since the primary purpose of the SPEREXSYS was to reduce ambiguity in the Voice Decoder's output, the testing philosophy was to carefully choose test cases which helped to measure the reduction of ambiguity due to imposing the constraint that all voice decoder output must form grammatically correct sentences.

In order to assist in this task, the 200 words which comprise the SPEREXSYS vocabulary were chosen so as to maximize ambiguity. For example, the words "two","too", and "to" were included because they sound identical even though they have completely different syntactic and semantic functions. Also, the words "peak" and "peek" (and "peaking" and "peeking") were chosen because they sound identical and have identical syntactic definitions. They can only be distinguished at the semantic level. Some of the words in the chosen vocabulary are fairly uncommon but were chosen because they sound quite similar to other more common words. The words "eunichs" and "units" are an example of this.

If a vocabulary had been chosen to reflect a set of words which have more common usage, there would be less ambiguity. It was therefore thought that the problems which would occur due to a vocabulary of a size significantly larger than 200 words could be simulated by choosing a set of 200 words with an uncommonly frequent degree of ambiguity.

Two levels of testing were required. The first was to verify that the program code functioned as anticipated (i.e. - to insure that there were no coding bugs). This testing was accomplished for the most part by the dynamic path testing of modules as each major functional area was written. This level of testing is not significant to the evaluation of the SPEREXSYS design and, for that reason, will not be discussed here.

The second level of testing was to evalute the SPEREXSYS design. This level of testing was concerned with answering such questions as:

1. How well does the SPEREXSYS handle identical sounding word ambiguities?
2. How does the SPEREXSYS respond when the correct word is not the highest probability word?
3. How much do changes in the search depth and acceptance threshold parameters affect the performance of the system?
4. How well does the SPEREXSYS find the end of sentences when the speaker is uttering consecutive sentences without stopping?
5. How effective is the SPEREXSYS at assisting the voice decoder to find correct word boundaries in connected speech?

6. How effective is the short term memory?

Rules of Testing

The test sentence was first diagrammed by dividiing it into phonemes on a time scale. The sentence "The peak got snow," is so diagrammed in this example:

(0) TH (5) E (15) P (20) EA (30) KG (35) O (45) T (50) S (55) N (60) O (70) FPUNCT (100).

The numbers in parentheses are Seelandt clock times. They mark the times of phoneme transition. The consonant phonemes are assigned a duration of five Seelandt time units. The vowel phonemes are assigned a duration of ten Seelandt time units. This was an arbitrary assignment scheme and is not significant other than it is a rough approximation of actual phoneme durations in normal speech. When it was possible to combine the terminal phoneme of one word with the initial phoneme of another word, it was done. For example, "peak" and "got" above, both share a common phoneme that sounds like the letter "g."

When the Voice Decoder is given the approximate time of the start of a next word, it may look back at most only one phoneme to begin looking for the start of the next word. The one exception to this is when the previous two phonemes were "s" and "t" such as at the end of the words "mast" or "missed." For example, when the words "missed stair" are pronounced in connected speech the "st" sounds may occur only once and be shared by both words.

When it was possible to insert bogus phonemes, they were inserted. An example of this is the pronunciation of the two words "go on." "Go" does not end with a "w" sound and "on" does not begin with a "w" sound, but when the two words are spoken together, a "w" sound occurs in the transition between the "o" in "go" and the "o" in "on."

The last rule is that no phonemes can be ignored from the start-word time given by the EPFE and the actual start of the word used by the voice decoder (with the exception of periods of silence - - FPUNCTS).

Test Number One

Purpose of Test: The purposes of this test are to examine the ability of the SPEREXSYS to:

1. Interpret a short single sentence.
2. Find word boundaries even when the boundary is a shared phoneme.
3. Respond accurately even when a wrong word is entered with a higher probability of likelihood than the correct word.
4. Distinguish between words with identical sound and identical syntactic definitions.

Test Specification: The sentence, "The peak got snow," will be input. The terminal phoneme of "peak" and the initial phoneme of "got" will be a shared "g" sound. The words "peak" and "peek" will both be entered with identical probabilities of likelihood for the word "peak." The word "no" will be entered with a higher probability of likelihood than the word "snow" for the last word in the sentence. Acceptthresh and searchdepth will be entered as .75 and 2 respectively.

Input Utterance: (0) TH (5) Ë (15) P (20) EA (30) G (35) Ŏ (45) T (50) S (55) N (60) Ō (70) FPUNCT (100).

Test Observations: The test observations are included in their entirety in appendix C.

Test Results and Conclusions:

1. The SPEREXSYS was able to properly interpret this sentence on the first attempt even with the ambiguities.
2. For this example, the SPEREXSYS was able to find the word boundaries between all four

words in the sentence even though the Voice Decoder was unable to do so.

3. Because of the syntactic constraint that does not allow the four words, "The peak got no," to be a complete sentence, the fact that the word "no" was entered with a higher probability of likelihood than the correct word "snow" did not cause the SPEREXSYS to fail to find the proper sentence as its first choice. Similar successful results would not be expected if a wrong word was given a higher probabiltiy of likelihood by the Voice Decoder if it fit syntactically into the sentence.

4. The Voice Decoder was unable to distinguish between the two nouns "peek" and "peak." The SPEREXSYS was unable to help in this ambiguity. This was expected because semantic information is necessary to make this decision. Acoustics and syntax are insufficient to make the proper distinction between the two words. The reason "The peak got snow" was output before "The peek got snow" was due only to the fact that "peak" was entered before "peek."

<u>Test Number Two</u>

Purpose of Test: The purposes of this test are to examine the ability of the SPEREXSYS to:

1. Find the end of sentences when the speaker is uttering consecutive sentences without stopping. (To make this test particularly difficult, no sentential pause will be inserted between the sentences).
2. Respond to improperly interpreted sentences.
3. Demonstrate improved performance with the use of the short term memory.

Test Specification: This test will be administered in three parts:

Part I: Input the sentence , "Your error was wrong." The word "air" will have higher probability of likelihood than the word "error."

Part II: The sentence is expected to fail the first time through the SPEREXSYS for reasons outlined in part 3 of the "Results and Conclusions" to test number one. Simulate a better pronunciation of the word "error" the second time through by giving it a higher probability of likelihood than "air." All other inputs will remain identical to the first test.

Part III: Properly input the sentence, "Their error was right," by inputting only the correct words each with a probability of likelihood of 90%. After the word "right" is entered, input the sentence in part I ("Your error was wrong.") without inputting an FPUNCT (sentential pause) after the word "right" in the first sentence. (This gets the word "error" in the short term memory.) The sentence, "Your error was wrong," should be entered exactly as it was the first time in part I (only the times will be shifted so that it will start after the word "right" in the first sentence). Acceptthresh and searchdepth should be set at .75 and 2 respectively for all parts of this test.

Input Utterances:

Part I: (0) Y (5) Ō (15) R (20) Ĕ (30) R (35) W (40) A (50) Z (55) R (60) Ŏ (70) N (75) G (80) FPUNCT (110).

Part II: (0) Y (5) Ō (15) R (20) Ĕ (30) R (35) Ō (45) R (50) W (55) A (65) Z (70) R (75) Ŏ (85) N (90) G (95) FPUNCT (125).

Part III: (0) TH (5) Ĕ (15) R (20) Ĕ (30) R (35) W (40) A (50) Z (55) R (60) Ŏ (70) Ē (80) T (85) Y (90) Ō (100) R (105) Ĕ (115) R (120) W (125) A (135) Z (140) R (145) Ŏ (155) N (160) G (165) FPUNCT (195).

Test Observations:

1. The SPEREXSYS failed to recognize the

sentence in part I.

2. The SPEREXSYS properly recognized the sentence on the first attempt in part II.
3. The SPEREXSYS properly recognized the sentence on the first attempt in part III.

Test Results and Conclusions:

1. The sentence, "Your error was wrong," fai: as expected the first time through because "air" was thought to be the correct word by the SPEREXSYS. Note that even perfect semantics at the sentence level would not have helped to find the correct word since the sentence, "Your air was wrong," is a sentence which is semantically correct all by itself. Semantics at a conversational level would be needed to determine which word made more sense within the scope of the conversation.
2. The first sentence was properly interpreted, as would have been expected, after the user followed the instructions of the SPEREXSYS to pronounce the word more clearly the second time.
3. In part III of this test, the end of the sentence was found after the next 2 words

had been looked at. It would have been discovered by looking at only the next word if acceptthresh had been set to anything between .83 and .9. If acceptthresh had been very low (around .2), it may have been that the end of the first sentence would not have been found. This leads to the conclusion that acceptthresh should be set as high as possible without eliminating the correct words.

4. The use of the short term memory, in part III, prevented the occurrence of the misinterpretation which happened in part I of this test.

Test Number Three

Purpose of Test: The purposes of this test are to examine the ability of the SPEREXSYS to:

1. Distinguish between long and short words with the same probabilities for the next word in the string.
2. Properly interpret paragraphs constructed out of long sentences uttered without stopping at the end of each sentence to insure the SPEREXSYS properly interpreted it.

Test Specification: The following paragraph will be uttered into the (simulated) Voice Decoder:

> The Airforce general was speaking to his staff about some recent C3 issues. He told us that there was nothing ambiguous about the intelligence report. The Army MI people and our own intel folks all agree. The enemy is running short on ammunition. We must have the communications to get this information out to all our units.

If the SPEREXSYS allows it, the words "Air Force" will be used as equal probability candidates for the word "Airforce." Similarly, the following sets will be used as equal probability candidates for the correct word:

sea cubed -- C^3

see cubed -- C^3

itch ewes -- issues

am big you us -- ambiguous

reap port -- report

arm me -- army

These are only a few of the equal probability and near equal probability words which are to be entered along with the correct words. Acceptthresh and searchdepth will be set to .75 and 2 respectively.

Input Utterances: (0) TH (5) Ē (15) Ĕ (25) R (30) F (35) Ō (45) R (50) S (55) G (60) Ĕ (70) N (75) R (80) A (90) L (95) W (100) A (110) Z (115) P (120) Ē (130) K (135) Ē (145) N (150) G (155) T (160) Ö (170) H (175) Ĭ (185) Z (190) T (195) Ă (205) F (210) A (220) B (225) OU (235) T (240) S (245) O (255) M (260) R (265) Ē (275) C (280) Ĕ (290) N (295) T (300) FPUNCT (330) C (335) K (340) Ū (350) B (355) D (360) Ĭ (370) SH (375) Ū (385) Z (390) FPUNCT (420) H (425) Ē (435) T (440) Ō (450) L (455) D (460) Ŭ (470) S (475) TH (480) Ĕ (490) R (495) W (500) A (510) Z (515) N (520) O (530) TH (535) Ē (545) N (550) G (555) Ă (565) M (570) B (575) Ĭ (585) G (590) Y (600) Ū (610) W (615) Ŭ (625) S (630) A (640) B (645) OU (655) T (660) TH (665) Ē (675) Ĭ (685) N (690) T (695) Ĕ (705) L (710) Ĭ (720) G (725) Ĕ (735) N (740) S (745) R (750) Ē (760) P (765) Ō (775) R (780) T (785) FPUNCT (815) TH (820) Ē (830) Ō (840) R (845) M (850) Ē (860) Ĕ (870) M (875) Ŏ (885) Ē (895) P (900) Ē (910) P (915) Ŭ (925) L (930) Ă (940) N (945) D (950) O (960) R (965) Ō (975) N (980) Ĭ (990) N (995) T (1000) Ĕ (1010) L (1015) F (1020) Ō (1030) K (1035) S (1040) Ŏ (1050) L (1055) A (1065) G (1070) R (1075) Ē (1085) FPUNCT (1115) TH (1120) Ē (1130) Y (1135) Ĕ (1145) N (1150) Ĕ (1160) M (1165) Ē (1175) Ĭ (1185) Z (1190) R (1195) Ŭ (1205) N (1210) Ē (1220) N (1225) G (1230) SH (1235) Ō (1245) R (1250) T (1255) Ŏ (1265) N (1270) Ă (1280) M (1285) Y (1295) Ū (1305) N (1310) Ĭ (1320) SH (1325) Ŭ (1335) N (1340) FPUNCT (1370) W (1375) Ē (1385) M

(1390) Ŭ (1400) S (1405) T (1410) FPUNCT (1440) H (1445) Ă (1455) V (1460) TH (1465) Ë (1475) C (1480) Ŏ (1490) M (1495) Y (1505) Ū (1515) N (1520) Ĭ (1530) C (1535) Ā (1545) S (1550) Ŭ (1560) N (1565) Z (1570) T (1575) Ū (1585) G (1590) Ĕ (1600) T (1605) TH (1610) Ĭ (1620) S (1625) Ĭ (1635) N (1640) F (1645) Ō (1655) R (1660) M (1665) Ā (1675) SH (1680) Ŭ (1690) N (1695) OU (1705) T (1710) Ū (1720) W (1725) Ŏ (1735) L (1740) Ŏ (1750) R (1755) Y (1760) Ū (1770) N (1775) Ĭ (1785) T (1790) S (1795) FPUNCT (1825).

Test Observations:

1. When the correct word was not the word with the highest likelihood for any word other than one of the last three words in the sentence, then the SPEREXSYS failed to properly interpret the sentence on the first attempt.
2. Eventually (see appendix B), all of the sentences in this paragraph were properly interpreted.

Test Results and Conclusions:

1. The introduction of the similar sounding word sets (prescribed in the Test Specification above) did not cause the SPEREXSYS to fail to properly identify the

correct sentences. In some instances, (i.e. - the substitution of all four words "am big you us" for "ambiguous" was not allowed due to syntactic constraint).

2. Essentially, the SPEREXSYS is making a decision on the third word back from the end of all strings. This is (by design) the nature of the deterministic decision making process in the SPEREXSYS. Semantic analysis during string construction is not yet employed in the SPEREXSYS. Acoustics and syntax are sometimes insufficient to find the correct identity of this third word back from the end of all strings. The result is that if the correct third word back does not have the highest word likelihood probability by the time it has been run through the short term memory and the longer word preference modules, it will be rejected. When this happens, the only recourse left to the SPEREXSYS is to ask the user to repeat the sentence and hope for better results on the next attempt. This inconvenience emphasizes the need for semantic analysis which has the effect of boosting the word probability of the correct word above all other word

probabilities. This must be done during the MAKEDECISION module function in the SPEREXSYS (i.e. - before the third words back go into the module to kill low probability strings).

V. Summary, Conclusions, and Recommendations

Subject to the accuracy of the acoustic analyzer and the accuracy and completeness of the English Parser, a near real time general solution to the application of syntactic constraints to spoken English recognition has been developed. This solution is functionally equivalent, in many ways, to the syntax processing of spoken English in the human brain. Because it closely models the syntax processing of the Human Speech Recognition System (HSRS), it would be most effective when used with the several levels of semantic analysis which are also evidently operational in the HSRS. Hence, it is a necessary part of the eventual general solution to the English speech recognition problem.

A. Summary and Conclusions

The purpose of this thesis was to find and develop a way to interface the Milne English Parser with the AFIT Voice Decoder so that the accuracy of the Voice Decoder would be improved by the additional constraint of requiring its output to form grammatically (syntactically) correct English sentences. It was thought that by so constraining the output of the Voice Decoder that additional information would be provided to help resolve Voice Decoder ambiguities such as finding word boundaries in connected speech and choosing between identical sounding words (such as "to", "too" and "two"). These ambiguities exist in that there is insufficient information in the acoustic data alone to resolve them. In addition, it was hoped that the application of these syntactic constraints might prove useful in the process of distinguishing between approximately likely options available to the Voice Decoder during its final decision making of the analysis of the input utterance. Contingent on the successful development of a solution to the above stated requirements, this thesis was to measure the degree of success achieved in each of the aforementioned requirements. In accomplishing all of these, it was desirable to investigate and develop a Voice Decoder - English Parser interface which functions similarly to the same acoustic analyzer - syntactic analyzer interface in the Human Speech Recognition System (HSRS).

All of this was accomplished. Some of the qualitative specifics of these accomplishments are discussed below:

1. The solution to this problem for the most part addresses the syntax aspects of the spoken English recognition problem. It includes some crude semantic analysis to help resolve some ambiguities left unresolved by the syntactic analysis. The solution is called the Spoken English Recognition Expert System (SPEREXSYS). Aside from the human interface aspects of operation (which were

due to implementation constraints) it is a near real time solution. It can be easily implemented as a real time solution by upgrading the hardware and communications with existing technology.

2. The SPEREXSYS was designed and developed to black box the type of acoustic analyzer (voice decoder) which is used. In other words, the SPEREXSYS operates independently from the type and design of the voice decoder (the voice decoder can be isolated or connected word, small or large vocabulary, any type of feature set extraction, any bit rate, any kind of word recognition scheme) as long as the voice decoder has the following attributes:

a. It can remember the input utterance.

b. It can determine the acoustic likelikhood of match of words in the vocabulary which might be the next word in the input string beginning at or around some specified time in the input utterance.

c. It can determine the start and stop times of every word which it determines to be a likely match.

3. It has been demonstrated that the solution to the problem of syntactically constraining acoustically analyzed speech must be deterministic in nature (meaning that it makes decisions one word at a time from left to right

without ever backtracking and with limited lookahead) in both electronic computers and the human brain. The SPEREXSYS is able to function psychologically equivalent to the syntactic analysis processing of the human brain. It also predicts the point at which semantic constraints should be introduced in order to maintain psychological compatibility with the semantic processes in the human brain. This was done by using the one buffer lookahead theory developed by Milne (Ref 17). It was decided to rely on the one buffer lookahead technique in order to assign syntactic functions to each word being considered, but that because of the increased confusion in speech (compared to written English) as to the location of word boundaries, it was decided to allow two buffers of lookahead before allowing for semantics to be introduced. This was thought to improve the probability of finding the right word before making a final decision as to the selection of a word based on semantics. It was also demonstrated that this final word selection (from the high probability options) must be made on the basis of semantics. In the SPEREXSYS, the final word selection should occur at the third word back from the end of a string under construction and also for the last two words in a sentence.

4. The SPEREXSYS incorporated functions which simulated the psychological functions (in the HSRS) of short term memory and longer word preference. More experimentation with the HSRS needs to be done in order to

more accurately describe (and apply in the SPEREXSYS) these functions.

5. The SPEREXSYS can find word boundaries, which the voice decoder cannot find, even when it is a shared phoneme (or set of phonemes) as long as the acoustic analyzer (voice decoder) is accurate enough to provide the right answer as one of the top few options and the English syntax is sufficient to resolve the ambiguities. Syntax was sufficient in most cases (during the testing of the SPEREXSYS), but in many others, semantics was necessary to resolve word boundary ambiguities.

6. The SPEREXSYS can distinguish between identically sounding words as long as the words have different syntactic functions. More specifically, the homonymns and crossonyms (identical sounding strings of phonemes) which are eliminated must form syntactically illegal or improbable strings. Semantics is required to distinguish between homonymns which have identical syntactic functions.

7. In the instances where the correct word was not identified as the most likely word by the voice decoder, the SPEREXSYS was able to choose the correct word if the words which were identified by the voice decoder as more likely words either did not fit syntactically in the sentence or led to improbable string constructions. This decision was much more likely to be made correctly if the voice decoder mistake was made within the last three words of a sentence (because syntactic constraints are much more

strict within the last three words of a sentence).

8. When the SPEREXSYS fails to properly interpret a sentence in its first attempt, it will output its best guess (sentence) and ask the user to repeat the sentence paying particular attention to the pronunciation of the words which were improperly interpretted the first time. The second attempt is usually more successful than the first.

9. The SPEREXSYS is able to properly interpret several sentences which are uttered in continuum without the user having to stop in between sentences to insure that the last sentence was properly interpretted.

10. The use of the short term memory which was modelled into the SPEREXSYS was helpful in increasing the accuracy of the SPEREXSYS in those instances when the wrong choice would have been made had it not been that the correct word was spoken in a previous sentence.

11. One of the user initialized (and program adjustable) variables which was the acceptance threshold (acceptthresh) should be set as high as possible without interfering with the selection of the correct words. This variable is useful in determining where the end of sentences are. The higher its value is, the earlier that determination can be properly made.

12. In addition to some of the above mentioned methods for increasing the effectiveness of the voice decoder, the SPEREXSYS improves the reliability of the voice decoder by

reducing the size of the vocabulary it has to search. This is done by applying the syntactic constraints to the next word in a string before the input utterance is analyzed and word options are considered (in the voice decoder). This has the effect of reducing the vocabulary size of possible words. Since the reliability of a voice decoder is related to its vocabulary size, this vocabulary reduction results in improved voice decoder reliability.

13. Although the SPEREXSYS is often forgiving if the correct word is not always the word chosen as most likely by the voice decoder, the SPEREXSYS is highly reliant on the voice decoder choosing the correct word as the most likely word most of the time.

B. Recommended Improvements and Enhancements

The following are areas which need to be improved, rethought, further researched, or enhanced:

1. More research needs to be done on the behavior, and effect of, the short term memory in the HSRS. The impact of the short term memory probably decreases with time perhaps with respect to the logarithm of the time since the utterance. It may favor certain types of words such as uncommon words, longer words, nouns and verbs,etc. It may be able to be influenced by semantics. These are things that need to be investigated and the results incorporated into the SPEREXSYS.

2. The phenomenon of favoring longer words over shorter words was the result of experimentation. This researcher is not convinced that the data gathered from these experiments conclusively demonstrate that this phenomenon is active in the manner in which it has been interpreted and applied in this thesis. More research needs to be done in this area.

3. After much rethinking of the basic assumptions that lead to the use of a two word buffer lookahead in this thesis, it appears that if semantics can be incorporated during string construction (and it should be) then only a one word buffer lookahead should be used. As it currently stands, the two word lookahead interferes with the mechanism that prefers longer words over shorter words.

4. The dynamic readjustment of the two variables searchdepth and acceptthresh should be further studied. If acceptthresh is dynamically readjusted, it should be adjusted based on the current track record of correct word likelihoods, not on a blind incrementing and decrementing algorithm. If searchdepth is readjusted, it may also be desirable to increase the number of buffers of lookahead. This recommendation may be withdrawn when semantic analysis during string construction becomes available.

5. The English Parser should be put on the VAX (as well as the voice decoder when it is ready) in order to eliminate the time consuming communications across a low speed modem link.

6. The current translation of English Parser feature types into a vocabulary list of possible next words from which the voice decoder can choose is a very inefficient and time consuming process. This process could perhaps be speeded up by the following:

a. Do a front end elimination of illegal feature types.

b. Do a check and elimination for reduntantly specified types.

c. Use parallel searching and processing of the translation of legal types into vocabulary lists.

7. Appendix D outlines a theory on the way the HSRS searches for best word matches which would eliminate the need for the entire function discussed in number 6 above.

8. When a real voice decoder is eventually hooked up to the front end of the SPEREXSYS, use a vocabulary size small enough to ensure that the correct word is the most likely word most of the time.

C. Possible Future Extensions of This Work

The syntactic constraint of the voice decoder's output is a critically important function the speech recognition process. But it is quite clear that syntax alone is inadequate to constrain the output of the voice decoder so that general English speech recognition can be accomplished. Several levels of semantic analysis have been suggested in this thesis. At least intra-sentential semantics must be developed and integrated into the SPEREXSYS.

Beyond that, an entire hierarchical system which is modelled after the HSRS needs to be developed. The SPEREXSYS might be useful (although this is not suggested by the discussion in appendix D) as the syntax analyzer in such a system since it behaves in a psychologically similar fashion to the HSRS.

Bibliography

1. Akmajan, A., and F. Heny. An Introduction to the Principles of Transformational Syntax. Cambridge, Mass: MIT Press, 1975.

2. Bach, Emmon. An Introduction to Transformational Grammars. New York: Holt, Rinehart and Winston, 1964.

3. Barr, Avron, and James Davidson. "Representation of Knowledge," Handbook of Artificial Intelligence, edited by Avron Barr and Edward A. Felgenbaum. DTIC document number AD A074078, 1980.

4. Chomsky, N. Aspects of the Theory of Syntax. Cambridge, Mass: MIT Press, 1965.

5. Chomsky, N. The Logical Structure of Linguistic Theory. New York: Pelnum Press, 1975.

6. Doddington, George R., and Thomas B. Schalk. "Speech Recognition, Turning Theory to Practice," IEEE Spectrum, 18:26-32 (September 1981).

7. Erman, Lee D. Fredrick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. "The Hearsay II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty." Computing Surveys, 12: 213-253 (June 1980).

8. Foderaro, John K., and Keith L. Sklower. The Franz Lisp Manual. Regents of the University of California, 1981.

9. Gardner, Anne. "Search," Handbook of Artificial Intelligence, edited by Avron Barr and Edward A. Felgenbaum, DTIC document number AD A074078, 1979.

10. Gardner, Anne,et al. "Natural Language Understanding," Handbook of Artificial Intelligence, edited by Avron Barr and Edward A. Felbenbaum. DTIC document number AD A076873, 1979.

11. Kabrisky, Matthew. A Proposed Model for Visual Information Processing in the Human Brain. Urbana: University of Illinois Press, 1966.

12. Kernigan, Brian W., and P. J. Plauger. "Programming Style: Examples and Counterexamples," Computing Surveys, Vol 6, No. 4-303-319 (December 1974).

13. Koutsoudas, A. Writing Transformational Grammars: An Introduction. New York: McGraw-Hill, 1966.

14. Lesser, Victor R., Richard D. Fennel, Lee D. Erman, and

D. Raj Reddy. "Organization of the Hearsay II Speech Understanding System," IEEE Transactions on Acoust., Speech, and Signal Processing, ASSP-23: 11-24 (February 1975).

15. Levinson, Stephen E., and Mark Y. Liberman. "Speech Recognition by Computer," Scientific American. New York: Scientific American, April 1981: 64-76.

16. Marcus, Mitchell P. A Theory of Syntactic Recognition for Natural Language. Cambridge, Mass: MIT Press, 1980.

17. Milne, Robert W. Handling Lexical Ambiguity in a Deterministic Parser. PhD Dissertation. University of Edinburgh, Edinburgh, Scotland. Not published.

18. Montgomery, Gerald J. Isolated Word Recognition Using Fuzzy Set Theory. MS Thesis GE/EE/82D-74. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December, 1982. DTIC document number AD-A124 851.

19. Peters, Lawrence J. Software Design: Methods and Techniques. New York: Yourdon Press, 1981.

20. Seelandt, Karl G. Computer Analysis and Recognition of Phoneme Sounds in Connected Speech. MS Thesis GE/EE/81D-53. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.

21. Stevens, W.P., et al. "Structured Design," IBM Systems Journal, Vol 13, No. 2 (1974).

22. Winston, Patrick Henry. Artificial Intelligence. Reading, Mass: Addison-Wesley, 1977.

APPENDIX A

SPEREXSYS PROGRAM LISTING

```
;*** SYSTEM FUNCTIONS DEFINED BY FRANZ LISP NOT IN FRANZ LISP ***

(declare (macros t))

(defun max (alist)
        (prog ()
                (return (maxel alist 0]
        (defun maxel (alist n)
                (cond
                    ((null alist) n)
                    (t (cond
                            ((greaterp (car alist) n)(maxel (cdr alist)
                                                        (car alist)))
                            (t (maxel (cdr alist) n]

;*** GENERAL PURPOSE, GENERAL USE FUNCTIONS ***

(defun printstglst (remstg)
        (cond
                ((null remstg) t)
                (t (terpr)(printstring (cdar remstg))(printstglst (cdr remstg]
                                        ;cdar remstg = words of first string
                                        ;i.e.- '(the (0 15) .95)' is a word

        (defun printstring (remstg)
                (cond
                  ((null remstg) nil)
                  ((equal (caar remstg) 'fpunct)(princ '| |))
                  (t (princ '| |)(print (caar remstg))(printstring (cdr remstg]
                                                ;caar remstg = word.dict
                                                ; i.e.- 'the' is a word.dict

;**************************************************************************
;***  THIS IS SPEREXSYS - THE SPOKEN ENGLISH RECOGNITION EXPERT SYSTEM  ****
;**************************************************************************



;**************************************************************************
;**************************************************************************
;**************************************************************************
;**************************************************************************

(defun spxsinit ()   ; mod 1 - called by sperexsys - initializes the sperexsys
                     ;                                              system.

(prog ()

(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)
(princ '|**********************************************************************|)
(terpr)
(princ '|***                                            ***|)
(terpr)
(princ '|***  Welcome to the SPoken English Recognition EXpert SYStem    ***|)
(terpr)
(princ '|***                  (SPEREXSYS)                      ***|)
(terpr)
(princ '|***                                            ***|)
(terpr)
(princ '|**********************************************************************|)
```

```
(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)
(princ '|Please ready the Milne English Parser and the AFIT Acoustic analyzer.|)
(terpr)
(princ '|When they have been readied, input the device I.D. of the |)(terpr)
(princ '|English Parser. It should be of the following form: /dev/ttyi7 |)
(terpr)(terpr)(terpr)(terpr)(terpr)(princ '|> |)
(setq epoutport (read 'epoutport))
(setq epoutport (outfile epoutport 'a))   ; open output port to DEC-10
(terpr epoutport)   ; this puts a vertical bar in the ep's output buffer.
                    ; it is necessary in order to properly trigger the reading
                    ; of the epresponse the first time.
(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)
(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)]

;*************************************************************************
;*************************************************************************
;*************************************************************************
;*************************************************************************


;*************************************************************************
;*************************************************************************
;*************************************************************************
;*************************************************************************

(defun global (i s m tim)  ;mod 2.2.1 - called by epfe - initializes global vbls
       (prog ()
              (load 'vocdict)   ; all the features (grammar types) defined
                                ; as a set of vocabulary words

              (load 'dict.spxs)   ; list of legal & illegal features

              (setq inittime tim)   ; start time for a sentence

              (setq topchoicenum i)   ; same as searchdepth in semanalyzer

              (setq numstrings (times i i))    ; number of strings allowed
                                               ; to be active in epfe

              (setq minaccept (times m 3))  ; 3 times the value of
                                            ; acceptthresh

              (setq init 1)   ; if = 1, then this is the first time through
                              ; for this sentence

              (setq maxwordtim 200)   ; approximate time it takes to pronounce
                                      ; longest English word

              (setq maxstnum s)   ; number of last used stringnum

              (setq nxgslst '())   ; epfe's request to voice decoder for
                                   ; next word in strings

              (setq optstglst '())   ; temporary stringlist value

              (setq optstg '())   ; used in mods 2.2.2.1.1 & 2.2.2.1.1.1

              (setq stringlist '())   ; the list of active strings

              (setq wordgslst '())          ; voice decoder's response to epfe's
                                            ; request for next words
```

```
(setq epreslst '())   ; English Parser response list

(setq topwordlst '())   ; list of top searchdepth words for
                        ; every active string

(setq optseslst '())   ; temporary variable used in manipulating
                       ; sentstlst

(setq sentstlst '()]   ; sentence string list (epfe's response
                       ; to the semantic analyzer)

;************************************************************************
;************************************************************************

;************************************************************************
;************************************************************************

(defun elim (stgnum stglst)   ; mod 2.2.2.1.1.1 - called by lookatnext2 -
                              ;                    eliminates nil epresponses
                              ;                    from epresponselist &
                              ;                    eliminates corresponding string
                              ;                    from stringlist
      (cond
            ((null stglst)(print '(err0r in elim)))
            ((eq (caar stglst) stgnum)
               ;caar stglst =  string number of first string in stringlist
                     (setq optstg (append optstg (cdr stglst))))
            (t (setq optstg (cons (car  stglst)(elim stgnum (cdr stglst]

;************************************************************************

(defun lookatnext2 (epres)   ; mod 2.2.2.1.1 - called by kilnilsts - idents nil
                             ;                 epresponses & calls elim
      (cond
            ((or (equal (cdr epres) '(nil))(null (cdr epres)))
                  (setq optstg '())(elim (car epres) epreslst)
                     ;optstg used for building new epreslst without
                     ;the nil responses
                  (setq epreslst optstg)(setq optstg '())
                  (elim (car epres) stringlist)(setq stringlist optstg]
                           ;here optstg used for building new stringlist
                           ;without strings that have corresponding nils
                           ;in epreslst

;************************************************************************

(defun kilnilsts (remstg)   ; mod 2.2.2.1 - called by formnxgs - eliminates
                            ;               strings from stringlist if correspond-
                            ;               ing epresponse is nil
      (cond
            ((null (car remstg)) t)
            (t (lookatnext2 (car remstg))(kilnilsts (cdr remstg]
                           ;car remstg =  first epres in remstg

;************************************************************************

(defun checkfpun (next2)   ; mod 2.2.2.2.1.1 - called by examnext2 - looks for
                           ;                   fpuncts in epres
      (cond
            ((null next2) nil)
               ;next2 is a list of a list of features in an epresponse
```

```
          ((equal (car next2) '(fpunc)) t)
          (t (checkfpun (cdr next2]

;****************************************************************************

(defun addtosestls (stglst stgnum)   ; mod 2.2.2.2.1.2 - called by examnext2 -
                                     ;                  adds strings to sentstlst

                                     ;                  which have fpunct in
                                     ;                  corresponding epres
     (cond
          ((null stglst)(princ 'Error in addtosesstls))
          ((equal (caar stglst) stgnum)   ; found string which matches
                                           ; epreslst for fpunc
               (setq sentstlst (cons (car stglst) sentstlst)))
                                           ; so add it to sentstlst
          (t (addtosestls (cdr stglst) stgnum]

;****************************************************************************

(defun examnext2 (epres)   ; mod 2.2.2.2.1 - called by fpunctproc - adds strings
                           ;                 to sentstringlist when fpuncts are
                           ;                 found in epreslst for that string
     (cond
          ((checkfpun (cdr epres))(addtosestls stringlist (car epres]
                           ;cdr epres = next2 = list of list of features from
                           ;                    English Parser

;****************************************************************************

(defun fpunctproc (remstg)   ; mod 2.2.2.2 - called by formnxgs - iteratively
                             ;               strips epres's from epreslst
     (cond   ; remstg = epreslst first time in
          ((null (car remstg)) t)
          (t (examnext2 (car remstg))(fpunctproc (cdr remstg]
                             ;car remstg is a complete epresponse for the
                             ;with the string number = caar remstg

;****************************************************************************

(defun gettim1 (words)   ; mod 2.2.2.3.1.2.1 - called by findtim1 - returns
                         ;                    tim2 for the last word in string
     (cond
          ((null (cdr words))(cadadar words))   ; if last word in string,
                                                ; return the termination
                                                ; time for that word
                                                ;                  (tim2)
          (t (gettim1 (cdr words]

;****************************************************************************

(defun translate (typen)   ; mod 2.2.2.3.1.1.1 - called by getnext1 - translates
                           ;                  parser's forms of types to voice
                           ;                  decoder's forms of types
     (prog ()
          (cond ((equal typen '(fpunct)) nil)   ; take out '[fpunc]',
               ((equal typen '(fpunc)) nil)    ; '[all]', and '[t]'
               ((equal typen '(t)) nil)        ; responses from
               ((equal typen '(all)) nil)      ; epresponses
          (t (return typen]
```

```
;**********************************************************************

(defun getnext1 (stgnum next2)  ; mod 2.2.2.3.1.1 - called by buildnxgs -
                                ;                 makes list of types (next1)
                                ;                 to be part of nextguess
     (cond
          ((null next2) '())
          (t
           (setq type (translate (car next2)))
           (cond
                ((equal type '(t)) '((all)))  ; this is here for
                                       ; future use when epfe won't be
                                       ; ignoring '[t]'
                (type (cons type (getnext1 stgnum (cdr next2))))
                (t (getnext1 stgnum (cdr next2]  ; these two lines are
                                            ; building translated next2
                                            ; (now called next1)

;**********************************************************************

(defun findtim1 (stgnum stglst)  ; mod 2.2.2.3.1.2 - called by buildnxgs -
                                 ;                 finds string in stringnum
                                 ;                 which matches stgnum and
                                 ;                 calls gettim1
     (cond
          ((null stglst)(terpr)(princ '|Error in findtim1 |)
          (princ '|for epres was nil for stgnum |)(print stgnum))
          ((equal (caar stglst) stgnum)(gettim1 (cdar stglst)))
                ;caar stglst = string number   cdar stglst = words
          (t (findtim1 stgnum (cdr stglst]

;**********************************************************************

(defun buildnxgs (stgnum next2)  ; mod 2.2.2.3.1 - called by makenxgslst -
                                 ;               builds next-guesses and adds
                                 ;               them to nextgslst
     (prog (type)
          (cond ((null next2) (return t)))
          (setq next1 (getnext1 stgnum next2))
          (setq tim1 (findtim1 stgnum stringlist))
          (setq nxgslst (cons (cons stgnum (cons tim1 next1)) nxgslst]
                ; now adds new nxgs to nxgslst
                ; form of nxgs is '(stgnum tim1 (feature set))'

;**********************************************************************

(defun makenxgslst (remstg)  ; mod 2.2.2.3 - called by formnxgs - makes
                             ;             next-guess-list for output to
                             ;             voice decoder. Iteratively strips
                             ;             epres's off of epreslst.
     (cond
          ((null remstg) t)  ; remstg = epreslst first time in
          (t (setq next1 '())(buildnxgs (caar remstg)(cdar remstg))
                              ;    string number  words
                                   (makenxgslst (cdr remstg]

;**********************************************************************

(defun formnxgs ()  ; mod 2.2.2 - called by epfe - forms next-guess-list
                    ;             (which is output to voice decoder).
     (prog ()
```

```
        (cond (init (setq nxgslst (list (cons maxstnum (cons inittime
                                                    '((all))))))
                 (+ 1 maxstnum)(setq stringlist (cons (list maxstnum)
                                                    stringlist)))
              (t (kilnilsts epreslst)
                 (fpunctproc epreslst)
                 (makenxgslst epreslst)))
        (terpr)(terpr)(princ'|On exiting formnxgs: nxgslst = |)
        (print nxgslst)(terpr]

;*****************************************************************************
;*****************************************************************************

;*****************************************************************************
;*****************************************************************************

(defun union2 (alist blist)   ; mod 2.2.3.1.1.1.1 - called by procfeatterm -
                              ;                     performs set union of two
                              ;                     lists
    (cond
        ((null alist) blist)
        ((member (car alist) blist)(union2 (cdr alist) blist))
        (t (cons (car alist)(union2 (cdr alist) blist]

;*****************************************************************************

(defun intersect2 (alist blist)   ; mod 2.2.3.1.1.1.2 - called by procfeatterm
                                  ;                     - performs set intersect
                                  ;                       of two lists
    (cond
        ((null alist) '())
        ((member (car alist) blist)
              (cons (car alist)(intersect2 (cdr alist) blist)))
        (t (intersect2 (cdr alist) blist)]

;*****************************************************************************

(defun compliment (universe nlist)   ; mod 2.2.3.1.1.1.3 - called by
                                     ;        procfeatterm - performs the set
                                     ;        compliment of nlist in the
                                     ;        given universe
    (cond
        ((null universe) '())
        ((member (car universe) nlist)
              (compliment (cdr universe) nlist))
        (t (cons (car universe)(compliment (cdr universe) nlist]

;*****************************************************************************

(defun procfeatterm (featterm)   ; mod 2.2.3.1.1.1 - called by getwordopt -
                                 ;                   interprets feature terms
                                 ;                   into word options from
                                 ;                   which the voice decoder
                                 ;                   can choose

  (cond
      ((null (cdr featterm))   ; featterm is a list of a single feature
         (cond
             ((member (car featterm) notfeatset) '())
             (t (eval (car featterm)))))
      ((equal (cadr featterm) 'or)   ; featterm is of form '(feature or -)'
```

```
        (cond
            ((atom (caddr featterm))
              (union2 (procfeatterm (list (car featterm)))
                      (procfeatterm (cddr featterm))))
            ((not (equal (caddr featterm) 'not))
              (union2 (procfeatterm (list (car featterm)))
                      (procfeatterm (caddr featterm))))
            ((atom (cadddr featterm))
              (union2 (procfeatterm (list (car featterm)))
                      (compliment all (procfeatterm (cdddr featterm
                                                                  )))))
            (t (union2 (procfeatterm (list (car featterm)))
                      (compliment all (procfeatterm (cadddr featterm
                                                      )))))))
      ((equal (cadr featterm) 'and)  ;featterm is of form '(feature and -)'
        (cond
            ((atom (caddr featterm))
              (intersect2 (procfeatterm (list (car featterm)))
                          (procfeatterm (cddr featterm))))
            ((not (equal (caddr featterm) 'not))
              (intersect2 (procfeatterm (list (car featterm)))
                          (procfeatterm (caddr featterm))))
            ((atom (cadddr featterm))
              (intersect2 (procfeatterm (list (car featterm)))
                          (compliment all (procfeatterm (cdddr featterm
                                                      )))))
            (t (intersect2 (procfeatterm (list (car featterm)))
                          (compliment all (procfeatterm (cadddr featterm
                                                      )))))))
      ((equal (car featterm) 'not)   ; the part of the previous featterm
                                     ; of the form '(not -)'
        (cond
            ((atom (cadr featterm))
                  (compliment all (procfeatterm (cdr featterm))))
            (t (compliment all (procfeatterm (cadr featterm))))))
      (t (terpr)(princ '|Error in procfeatterm: |)(terpr)(princ '|    |)
       (print featterm)
       (princ '|is not a legal feature-type from the Parser.|) '()]

;**********************************************************************

(defun getwordopts (featlist)   ; mod 2.2.3.1.1 - called by printwordopts -
                                ;               functions as a driver for
                                ;               the feature list interpreter
     (cond
          ((null featlist) '())   ; featlist = next1 first time in
          (t (union2 (procfeatterm (car featlist))
                                        ;car = first feature in featlist
                      (getwordopts (cdr featlist))]

;**********************************************************************

(defun printwords (wordopts)   ; mod 2.2.3.1.2 - called by printwordopts -
                               ;               prints word options out to
                               ;           user (11 per line)
     (prog (n)
        loopouter
        (setq n 11)
        (terpr)
        loopinner
```

```
(cond
      ((null wordopts)(return t))
      (t (princ '| |)(print (car wordopts))
                              ; first word in wordopts is printed
    (setq wordopts (cdr wordopts))(setq n (sub1 n))
                         ; the rest of the words are sent back
                         ; through again
       (setq wordcount (add1 wordcount))
       (cond
           ((plusp n)(go loopinner))   ; if 11 words have been
                                       ; printed on this line, start
                                       ; another line
           (t (go loopouter)))]

;**********************************************************************

(defun printwordopts (epresponse)   ; mod 2.2.3.1 - called by intefvocdec -
                                    ;         functions as a driver for
                                    ;              the user's listing of the
                                    ;         words from which the voice
                                    ;         decoder can choose
    (cond
      ((null epresponse)(terpr)(terpr)(princ '|Error: In printwordopts|))

      (t (terpr)(terpr)
         (princ '|****************************************************************|)
         (terpr)
         (princ '|Possible words for voice decoder to choose from are:|)
         (terpr)(cond
               ((equal (caaddr epresponse) 'all)(princ '|ALL WORDS|))

               (t (setq wordcount 0)   ; initialize wordcount which
                                       ; is incremented every time a
                                       ; word is printed in printwords
                 (printwords (getwordopts (cddr epresponse)))
                                          ;cddr epresponse = next1
               (terpr)(terpr)
            (princ '|   TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR|)
               (princ '|THIS OPTION FROM 200 TO |)
               (print wordcount)))   ; wordcount now = total
                                       ; number of words printed
         (terpr)
         (princ '|****************************************************************|)]


;**********************************************************************

(defun interfvocdec (remstg)   ; mod 2.2.3 - called by epfe - read print
                               ;         statements in this module for an
                               ;         explanation of its function
    (cond   ; remstg = nxgslst first time in
         ((null remstg) t)
         (t
          (terpr)(princ '|Please type in the voice decoder's response |)
          (princ '|to the following next-guess-request.|)
          (terpr)(princ '|Remember to use the following format:|)
          (terpr)(princ '|(stringnum (dict.name1 (tim1 tim2) prob)|)
          (princ '|(dict.name2 (tim1 tim2) prob)...)|)
          (printwordopts (car remstg))
                        ;car remstg is first next-guess-request (for
```

```
                        ;a single string) in remstg
            (terpr)(princ '|Next-guess-request = |)(print (car remstg))
            (terpr)(princ '|> |)
            (setq wordgslst (cons (cons (caar remstg)(cdr (read 'words)))
                                                          wordgslst))
                               ;caar remstg is string number
            (interfvocdec (cdr remstg)]

;*************************************************************************
;*************************************************************************

;*************************************************************************
;*************************************************************************

(defun findfpunct (remstg)   ; mod 2.2.4.1.1 - called by procwdgs - returns
                             ;                 word value for fpunct.
      (cond   ; remstg =  words in wdgs
            ((null remstg) nil)
            ((equal (caar remstg) 'fpunct)(car remstg))
                  ;caar remstg =  word.dict
                  ;searches every word.dict in remstg until it finds
                  ;an 'fpunct' or it returns nil
            (t (findfpunct (cdr remstg]

;*************************************************************************

(defun augsenstg (stgnum fpunval remstg)   ; mod 2.2.4.1.2 - called by procwdgs
                                           ;         - adds fpunct word
                                           ;           value to end of approp-
                                           ;           riate string in
                                           ;           sentstringlist

      (cond   ; fpunval is of the form '(fpunct (tim1 tim2) prob)'
            ; remstg is sentstlst first time in

            ((null remstg) t)
            ((equal (caar remstg) stgnum)
                  ;caar remstg =  stgnum of first sentence in remstg
                  (setq newstg (append (car remstg)(list fpunval)))
                  (setq optseslst (cons newstg optseslst))
                  (setq optseslst (append optseslst (cdr remstg))))
            (t (setq optseslst (cons (car remstg) optseslst))
              (augsenstg stgnum fpunval (cdr remstg]

;*************************************************************************

(defun calcnewprob (timsnprob)   ; mod 2.2.4.1.3.1 - called by changeprob - does
                                 ;                   new prob calculation.
      (prog (tim1 tim2 prob ans)
            ;returns prob +  (1-prob)[1/2(tim2 - tim1)/maxwordtime]
            (setq tim1 (caar timsnprob))
            (setq tim2 (cadar timsnprob))
            (setq prob (cadr timsnprob))
            (setq ans (diff 1.0  prob))
            (setq ans (times ans (diff tim2 tim1)))
            (setq ans (quotient ans 2.0 maxwordtim))
            (return (add prob ans]

;*************************************************************************

(defun changeprob (words)   ; mod 2.2.4.1.3 - called by procwdgs - changes prob
```

```
                      ;                   of word according to its wordlngth

   (cond
       ((null words)optseslst)   ; when all done, returns sentstlst
                                  ; with changed word probs
       (t (setq newprob (list (calcnewprob (cdar words))))
                                      ;cdar words =  times and
                                      ;prob for first word
         (setq newword (append (cons (caar words)(list (cadar words)))
                                  ;caar words =  word.dict for
                                  ;          first word
                                  ;cadar words =  '(tim1 tim2)'
                                  ;        for first word
                                                newprob))
         (setq optseslst (cons newword optseslst))
         (changeprob (cdr words]

;******************************************************************************

(defun getproblst (words)   ; mod 2.2.4.1.4 - called by procwdgs - makes a list
                            ;          of all the probs in wordguess
   (cond
       ((null words) '())
       (t (cons (caddar words)(getproblst (cdr words]
              ;caddar words is new prob for word

;******************************************************************************

(defun newalist (num alist)   ; mod 2.2.4.1.5.1 - called by orderlist - deletes
                              ;          first occurrence of num in
                              ;          alist, then returns alist.
                              ;          NOTE - THIS IS A LATE DESIGNED
                              ;          MODULE AND DOES NOT APPEAR IN
                              ;          IN THE THESIS CHARTS OR
                              ;          NARRATIVE.
   (cond
       ((null alist)(terpr)(princ '|Error in newlist: No match found.|) '())
       ((equal (car alist) num)(cdr alist))
       (t (cons (car alist)(newalist num (cdr alist]

;******************************************************************************

(defun orderlist (alist number)   ; mod 2.2.4.1.5 - called by procwdgs and by
                                  ;          choptomns (2.2.6.1) and by
                                  ;            ranksents (2.3) - orders
                                  ;            the top number of elements
                                  ;            in alist in decreasing order

   (prog (nextnum)
       (setq number (- number 1))
       (cond
           ((or (minusp number)(null alist))(return '()))
           (t (setq nextnum (max alist))
             (setq alist (newalist nextnum alist))
             (return (cons nextnum (orderlist alist number]

;******************************************************************************

(defun topfunct (prob words)   ; mod 2.2.4.1.6.1.1 - called by gettopwds - pulls
                               ;          out words with probs matching
                               ;          prob.
```

```
        (cond
                ((null words) '())
                (t (cond
                        ((equal prob (caddar words))
                                        ;caddar words = new prob for first word
                          (cons (car words)(topfunct prob (cdr words))))
                        (t (topfunct prob (cdr words]

;***************************************************************************

(defun gettopwds (problist n words)   ; mod 2.2.4.1.6.1 - called by findtopwds -
                                      ;               makes list of words in
                                      ;                       wordguess which match
                                      ;                       the probs in problist.
        (prog ()
                (setq n (- n 1))
                (cond   ; only do this for the top n probs in problist
                        ((minusp n)(return '()))
                        (t (return (append (topfunct (car problist) words)
                                        (gettopwds (cdr problist) n words]

;***************************************************************************

(defun striptopn (alist n)   ; mod 2.2.4.1.6.2 - called by findtopwds - keeps
                             ;                  only the top n words of alist.
        (prog ()
                (setq n (- n 1))
                (cond   ; this used to be necessary when gettopwds functioned
                        ; differently -- now its redundant
                        ((or (null alist)(minusp n))(return '()))
                        (t (return (cons (car alist)(striptopn (cdr alist) n]

;***************************************************************************

(defun findtopwds (problist n words)   ; mod 2.2.4.1.6 - called by procwdgs -
                                       ;               main driver for submods
                                       ;               which find top prob
                                       ;               words (n of them).
        (prog ()
                (return (striptopn (gettopwds problist n words) n]

;***************************************************************************

(defun procwdgs (wdgs)   ; mod 2.2.4.1 - called by dectopwds - picks top words
                         ;                  and makes topwordlst
        (prog (n fpunval)
                (setq fpunval (findfpunct (cdr wdgs)))
                                        ;cdr wdgs = words
                (cond (fpunval (augsenstg (car wdgs) fpunval sentstlst)
                                        (setq sentstlst optseslst)))
                                        ;car wdgs = string number
                (setq wdgs (cons (car wdgs)(shorttermmem (cdr wdgs))))
                        ; send all the words to the short term memory to
                        ; increase probs of words recently spoken
                (setq optseslst '())
                (setq wdgs (cons (car wdgs)(changeprob (cdr wdgs))))
                        ; send all words to changeprob to increase word probs
                        ; of longer words
                (setq optseslst '())
                (setq problist '())
                (cond ((null (cdr stringlist))(setq n numstrings))
```

```
                        ; if this is the first time through for this sentence,
                        ; allow for a greater margin of error for first words
                     (t (setq n topchoicenum)))
                  (setq topprlst (orderlist (getproblst (cdr wdgs)) n))
                  (setq topwordlst (cons (cons (car wdgs)(findtopwds topprlst n
                                                      (cdr wdgs))) topwordlst]

;****************************************************************************

(defun dectopwds (remstg)   ; mod 2.2.4 - called by epfe - iteratively strips
                            ;           wordguesses off of wordgslst (which
                            ;           is remstg in this mod) and sends them
                            ;           to procwdgs.
      (cond
            ((null remstg)(terpr)(terpr)(princ '|On exiting dectopwords: topwordlst = |)(print topwordlst))
            ((or (null (cdar remstg))(equal (cdar remstg) '(nil)))(dectopwds (cdr remstg)))
                  ;cdar remstg = words of the first wdgs in remstg

            (t (procwdgs (car remstg))(dectopwds (cdr remstg]

;****************************************************************************
;****************************************************************************

;****************************************************************************
;****************************************************************************

(defun getwords (stgnum remstg)   ; mod 2.2.5.1.1.1 - called by findwdsmatch -
                                  ;            returns the list of words
                                  ;            in topwordlst which have
                                  ;            same string number as
                                  ;            string being processed in
                                  ;            findwdsmatch.
      (cond   ; remstg = top word list first time in
            ((null remstg)(princ '|Error: no match in getwords.|))
            ((equal (caar remstg) stgnum)(cdar remstg))
                  ;caar remstg = string number, cdar = words
            (t (getwords stgnum (cdr remstg]

;****************************************************************************

(defun makests (string words)   ; mod 2.2.5.1.1.2 - called by findwdsmatch -
                                ;            throws away fpuncts. Also
                                ;            makes new strings with new
                                ;            words and concats to optstglst
      (cond
            ((null words) t)
            (t (cond
               ((equal (caar words) 'fpunct)(makests string (cdr words)))
                     ;caar words = word.dict of first word in words
                  (t (setq maxstnum (+ 1 maxstnum))
                        ; get the next unused string number
                  (setq optstglst (cons (append (cons maxstnum (cdr string

                        ))(list (car words))) optstglst))))
                        ; make new complete string and add it to
                        ; optstglst
                  (makests string (cdr words]

;****************************************************************************

(defun findwdsmatch (string)   ; mod 2.2.5.1.1 - called by newstrings - calls
```

```
                                        ;        makests with a string and its
                                        ;        associated top words
     (prog (words)
           (setq words (getwords (car string) topwordlst))
                                          ;car string = string number
           (makests string words]

;***********************************************************************

(defun newstrings (remstg)   ; mod 2.2.5.1 - called by startnsts - calls
                             ;               findwdsmatch with next string till
                             ;               stringlist is exhausted
     (cond   ; remstg is stringlist first time in
           ((null remstg) optstglst)
           ( t (findwdsmatch (car remstg))
                                      ;car remstg is first string in remstg
             (newstrings (cdr remstg]

;***********************************************************************

(defun builddeclst (remstg)   ; mod 2.2.5.2.1 - called by makedecision - builds
                              ;                 list of third words from end of
                              ;                 strings
     (cond
        ((null remstg) nil)
        (t (setq decword (caaddr (reverse (car remstg))))
                              ;caaddr is third word.dict from end of string
           (cond
              ((member decword declist) t)
              (t (setq declist (cons decword declist))))
           (builddeclst (cdr remstg)]

;***********************************************************************

(defun makedecision ()   ; mod 2.2.5.2 - called by startnsts - displays list of
                         ;          all third words from end of strings
                         ;          which killowsts will make decision on
                         ;          next
     (prog ()
       (cond ((cdddar stringlist)   ; only do this if there are more than
                                    ; two words in each string
         (setq declist (cons (caaddr (reverse (car stringlist))) '()))
         (cond
             ((not(equal declist '(nil)))(builddeclst (cdr stringlist))
                         ;declist is a list of all third word.dicts from
                         ;the end of every active string
                   (terpr)
                   (terpr)(princ '|A decision is now being made on the |)
                   (princ '|third word from the end of all strings.|)
                   (terpr)(princ '|The choices are:|)(terpr)
                   (print declist)(terpr)]

;***********************************************************************

(defun startnsts ()   ; mod 2.2.5 - called by epfe - initializes global strings
                      ;          used and calls newstrings.
     (prog ()
           (setq optstglst '())
           (setq stringlist (newstrings stringlist))
           (setq topwordlst '())
           (setq wordgslst '())
```

```
                (terpr)(princ '|After exiting startnsts: stringlist = |)
                (print stringlist)
                (makedecision)]

;**************************************************************************
;**************************************************************************

;**************************************************************************
;**************************************************************************

(defun calcstgprob (words)   ; mod 2.2.6.1.1.1.1 - called by stgprob - returns
                             ;                 cummulative of word probs in
                             ;                 string (just words here - no
                             ;                 stringnum)
        (cond
                ((null words) 0)
                (t (add (caddar words)(calcstgprob (cdr words]
                        ;caddar words =  prob of first word in words

;**************************************************************************

(defun stgprob (words)   ; mod 2.2.6.1.1.1 - called by getstgprobs -gets stgprob
                         ;                 and concats it problist and then
                         ;                 returns stgprob.
        (prog (prob)
                (setq prob (calcstgprob words))   ; get the string prob
                (setq problist (cons prob problist))   ; add it to problist
                (return prob]

;**************************************************************************

(defun getstgprobs (remstg)   ; mod 2.2.6.1.1 - called by choptomns and by
                              ;                 ranksents (2.3) - makes new
                              ;                 list of strings with stringprob
                              ;                 concatted on front of each strg
        (cond   ; remstg =  stringlist first time in
                ((null remstg) '())  ; remstg =  stringlist first time in
                (t (cons (cons (stgprob (cdar remstg))(car remstg))
                                        ;cdar remstg =  words of first string
                        ; the above adds stgprob to front of each string
                         (getstgprobs (cdr remstg]

;**************************************************************************

(defun gettopsts (prob remstg)   ; mod 2.2.6.1.2 - called by choptomns - returns
                                 ;                 the list of strings which
                                 ;                 have stringprobs above or
                                 ;                 equal to last prob in
                                 ;                 problist
        (cond   ; remstg =  stringlist first time in
                ((null remstg) '())
            ((minusp (diff (caar remstg) prob))(gettopsts prob (cdr remstg)))
                        ;caar remstg =  string prob for first string in
                        ;                                              remstg
                (t (cons (cdar remstg) (gettopsts prob (cdr remstg]

;**************************************************************************

(defun choptomns (remstg)   ; mod 2.2.6.1 - called by killowsts and by ranksents
                            ;                 (mod 2.3) - returns the
                            ;                 the top maxstgnum strings in
```

```
                                 ;                    stringlist
        (prog ()   ; remstg =  stringlist
                (setq remstg (getstgprobs remstg))
                (setq problist (orderlist problist numstrings))
                (return (gettopsts (car (reverse problist)) remstg]
                                        ;car =  lowest acceptable string prob

;*****************************************************************************

(defun overmin (words n)   ; mod 2.2.6.2.1.1 - called by checkminpr - returns
                           ;                     cummulative addition of last 3
                           ;                     word probs.
        (cond
                ((zerop n) 0)   ; quit when n (= 3) words have been processed
                ((null words)(add 1.0 (overmin words (diff n 1))))
                (t (add (caddar words)(overmin (cdr words)(- n 1]
                        ;caddar words =  word prob for first word in words

;*****************************************************************************

(defun checkminpr (words)   ; mod 2.2.6.2.1 - called by elimminacc - returns
                            ;                   a t if last three probs (added)
                            ;                   are greater than minaccept.
        (cond ((greaterp (overmin words 3) minaccept) t]   ; send back 't' only
                                    ; if last three word probs are above minaccept

;*****************************************************************************

(defun elimminacc (remstg)   ; mod 2.2.6.2 - called by killowsts - returns all
                             ;                 strings with last 3 probs above
                             ;                 minaccept threshold.
        (cond
                ((null remstg) '())
                (t (setq string (checkminpr (reverse (cdar remstg))))
                  (cond
                        ((null string) (elimminacc (cdr remstg)))  ; string not
                                        ; included in new stringlist if it did
                                        ; not pass test in checkminpr
                        (t (cons (car remstg)(elimminacc (cdr remstg]

;*****************************************************************************

(defun killowsts ()   ; mod 2.2.6 - called by epfe - driver for functions
                      ;               chop stringlist entries to numstrings and
                      ;               eliminates strings below minaccept threshold
        (prog ()
                (setq problist '())
                (setq stringlist (choptomns stringlist))
                (setq stringlist (elimminacc stringlist))
                (terpr)(princ '|After exiting killowsts: stringlist = |)
                (print stringlist)
                (terpr)(terpr)(princ '|To summarize the above stringlist, |)
                (princ '|the following strings are still active:|)
                (terpr)(printstglst stringlist)
                (cond (sentstlst (terpr)(terpr)(princ '|And the following |)
                                (princ '|sentences are to be forwarded to the|)
                                (princ '|semantic analyzer:|)
                                (printstglst sentstlst)]

;*****************************************************************************
;*****************************************************************************
```

```
;*******************************************************************
;*******************************************************************

(defun stgprint (string)  ; mod 2.2.7.1.1 - called by interfep - builds
                          ;               compacted strings with pause and
                          ;               xmts them to english parser
        (cond   ; string =  words first time in
                ((null string)(princ '|pause]).
                |)(drain)(princ '|pause]).
|epoutport)(drain epoutport))
                (t (print (caar string))(print (caar string) epoutport)
                          ;caar string =  first word.dict in string
                  (princ '| |)
                  (princ '| |epoutport)(stgprint (cdr string]

;*******************************************************************

(defun evals (instrs)   ; mod 2.2.7.1.2.1 - called by interfsem - evaluates
                        ;                   instrs
        (cond
                ((null instrs) t)
                (t (eval (car instrs))(evals (cdr instrs]

;*******************************************************************

(defun errorrecovry ()   ; mod 2.2.7.1.2 - called by interfep - receives
                         ;                 instructions for recovery and calls
                         ;                 evals to have them executed
        (prog (instrs)
                (terpr)(princ '|Please type in instructions to be evaluated.|)
                (terpr)(print '(remember to nest list them))(terpr)
                (princ '|> |)(setq instrs (read 'instrs))
                (evals instrs)]

;*******************************************************************

(defun interfep (string)  ; mod 2.2.7.1 - called by iteprest - sends compacted
                          ;                string with pause to english parser
                          ;                and builds a list of ep responses
        (prog (next2 epres char)
                (terpr)(terpr)
                (terpr)(princ '|Data from epfe to english parser follows:|)
                (terpr)(princ '|go1([|)(princ '|go1([|epoutport)
                (stgprint (cdr string))
                          ;cdr string =  words in string (no string number)
                loop2
                (setq next2 (readc))
                (print next2)(drain)
                (cond ((not (equal next2 '))(go loop2)))
                       ; read next2 only after '| has been read
                (setq next2 (read))(print next2)(terpr)(terpr)
                   ;next2 is a list of a list of possible features for the
                   ;next word in this string -- it is sent by the Parser
                (setq epres (cons (car string) next2))
                (setq epreslst (cons epres epreslst))]

;*******************************************************************

(defun iteprest (remstg)  ; mod 2.2.7 - called by epfe - iteratively strips
                          ;             strings from stringlist and calls interfep
        (cond   ; remstg =  stringlist first time in
```

```
            ((null (car remstg))
                ;car remstg = first string in remstg
            (setq piport 'stdin)(drain epoutport)   ; reset primary
                                        ; input port to the user's terminal
                  (terpr)(print '(epreslst is as follows:))
                  (terpr)(print epreslst)
                  (terpr)(terpr)
                  (princ '|Would you like to try the EP interface again?|)
                  (terpr)
                  (princ '|(r - rerun; i - new instrs; g - keep going)|)
                  (terpr)(princ '|> |)
                  (setq char (read))
                  (cond
                    ((equal char 'g) t)
                    ((equal char 'r)
                                    (setq epreslst '())
                                    (setq piport (infile '/dev/tty12))
                                    (drain epoutport)(iteprest stringlist))
                    ((equal char 'i)(terpr)(terpr)(terpr)
                          (princ '|!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!|)
                          (terpr)(princ '|You are entering very dangerous territory!  For assistance|)
                          (errorrecovry)
                          (iteprest '()))
                    (t (terpr)(print char)(princ '| is not a legal response.  The question was:|)
                          (iteprest '())))))

            (t (interfep (car remstg))(iteprest (cdr remstg]

;*************************************************************************
;*************************************************************************

;*************************************************************************
;*************************************************************************
;*************************************************************************

(defun epfe (i s m tim)  ; mod 2.2 - called by semanalyzer - functions as the
                         ;     interface between the english parser and the
                         ;     voice decoder.  Deterministically builds
                         ;     syntactically correct strings from the voice
                         ;     decoder's output and returns completed
                         ;   sentences to semanalyzer for semantic analysis.
  (prog ()
        (global i s m tim)  ; mod 2.2.1
        loop
        (formnxgs)  ; mod 2.2.2
        (terpr)(princ '|Output from epfe to voice decoder follows:|)
        (interfvocdec nxgslst)   ; mod 2.2.3
        (terpr)(princ '|This concludes output (next-guess-requests) |)
        (princ '|from the epfe to the voice decoder.|)(terpr)(terpr)
        (terpr)(princ '|Before entering dectopwds: wordgslst = |)
        (print wordgslst)
        (dectopwds wordgslst)   ; mod 2.2.4
        (cond ((null (car topwordlst))(return)))
        (setq nxgslst '())
        (startnsts)  ; mod 2.2.5
        (cond ((null stringlist)
                          (terpr)(princ '|Epfe done. Returning to semantic |)
                                         (princ '|analyzer. |)(return)))
        (killowsts)  ; mod 2.2.6
        (cond ((null stringlist)
                          (terpr)(princ '|Epfe done. Returning to semantic |)
```

```
                                   (princ '|analyzer. |)(return)))
        (setq init nil)  ; if its made it this far, it is no longer processing
                         ; the first word in the string.
        (setq epreslst '())
        (setq piport (infile '/dev/tty12))(drain epoutport)  ; set primary
                                         ; input port to the DEC-10 modem
        (iteprest stringlist)  ; mod 2.2.7
        (go loop)]

;**********************************************************************
;**********************************************************************
;**********************************************************************


;**********************************************************************
;**********************************************************************
;**********************************************************************
;**********************************************************************

(defun semaninit ()   ; mod 2.1 - called by semanalyzer - initializes semantic
                      ;               analyzer global variables.
        (prog ()
                (setq shortermem '())
                (terpr)(terpr)(terpr)
                (princ '|Control has now been turned over to the semantic |)
                (princ '|analyzer.|)(terpr)
                (princ '|This is the highest level of decision making in the |)
                (princ '|speech recognition process.|)(terpr)
                (princ '|In order to initialize the system error parameters, |)
                (princ '|please answer the following questions.|)(terpr)(terpr)
                (princ '|How many words deep will the Acoustic Analyzer have |)
                (princ '|to go in order to |)(terpr)
                (princ '|guarantee that the correct word will be recognized?  |)
                (princ '|(Normally this is "3").|)(terpr)(princ '|> |)
                (setq searchdepth (read 'searchdepth))(terpr)
                (princ '|What is the minimun acceptable average probabilty of |)
                (princ '|correctness for the last |)(terpr)
                (princ '|three words in a string?  (Normally this is .75).|)
                (terpr)(princ '|> |)(setq acceptthresh (read 'acceptthresh))
                (terpr)(terpr)(terpr)(terpr)
                (setq sentstart '1000)
                (setq inittim 0)]

;**********************************************************************
;**********************************************************************


;**********************************************************************
;**********************************************************************

(defun modfpuncts (sentstg)   ; mod 2.3.1.1 - called by incrfpuncts - if last
                              ;                 word int the sentence is an
                              ;                 fpunct, this mod adds 100 to its
                              ;                 word prob.
        (prog (newprob newword)
                (setq sentstg (reverse sentstg))
                (cond
                    ((equal (caar sentstg) 'fpunct)
                            ;caar sentsts = word.dict of last word in string
                        (setq newprob (add 100 (caddar sentstg)))
                                              ;caddar sentstg = prob of fpunct
```

```
                (setq newword (reverse (car sentstg)))
                (setq newword (reverse (cons newprob (cdr newword))))
                (return (reverse (cons newword (cdr sentstg)))))
             (t (return (reverse sentstg]

;*************************************************************************

(defun incrfpuncts (remstg)   ; mod 2.3.1 - called by ranksents - counts
                              ;              number of sentences in sentstlst
                              ;              and calls modfpuncts for each
                              ;              sentence.
      (cond  ; remstg = sentstlst first time in
             ((null remstg) '())
             (t
               (setq numstrings (add 1 numstrings))
               (cons (modfpuncts (car remstg)) (incrfpuncts (cdr remstg]

;*************************************************************************
;*************************************************************************


;*************************************************************************
;*************************************************************************

(defun newseslst (prob remstg)   ; mod 2.3.2.1.1 - called by topsent - sets
                                 ;              topsentence = sentence when
                                 ;              stgprob = prob, removes it from
                                 ;              sentstlst and returns sentstlst
      (cond
             ((null remstg)(terpr)(princ 'Error in newseslst.))
             ((equal (caar remstg) prob)(setq topsentence (cdar remstg))
                                         (cdr remstg))
                  ;caar remstg = sentence string prob
                  ;cdar remstg = sentence without string prob
             (t (cons (car remstg)(newseslst prob (cdr remstg]

;*************************************************************************

(defun topsent (prob)   ; mod 2.3.2.1 - called by ordersentlst - returns top
                        ;              prob sentence after removing it from
                        ;              sentstlst
      (prog ()
             (setq sentstlst (newseslst prob sentstlst))
             (return topsentence]

;*************************************************************************

(defun ordersentlst (orderedprobs)   ; mod 2.3.2 - called by ranksents - rank
                                     ;              orders sentstlst in decreasing
                                     ;              order by stgprobs.
      (cond
             ((null orderedprobs) '())
             (t (cons (topsent (car orderedprobs))(ordersentlst
                                                   (cdr orderedprobs]
                        ;car orderedprobs is highest prob in list

;*************************************************************************

(defun ranksents ()   ; mod 2.3 - called by semanalyzer - builds list of
                      ;              sentences ordered from highest to lowest
```

```
;                    choices based on semantic best sentence rules.

        (cond
                ((or (equal sentstlst '(nil))(null sentstlst)) nil)
                (t
                  (setq numstrings 0)
                  (setq sentstlst (incrfpuncts sentstlst))
                  (setq problist '())
                  (setq sentstlst (getstgprobs sentstlst))
                  (ordersentlst (orderlist problist numstrings]

;******************************************************************
;******************************************************************


;******************************************************************
;******************************************************************

(defun outrestsent (remstg)   ; mod 2.4.1 - called by printsent - outputs rest
                              ;                  of sentence (best guess) to user.
        (cond   ; remstg =  words of sentence first time in
                ((null remstg)(princ '| |))
                ((equal (caar remstg) 'fpunct)(princ '| |))
                        ;caar remstg =  word.dict of first word in sentence
                (t (princ '| |)(print (caar remstg))
                  (setq tempshortermem (cons (caar remstg) tempshortermem))
                        ; add this word temporarily to short term memory
                        ; if user approves sentence, it will be permanent,
                        ; otherwise, it will be forgotten
                  (outrestsent (cdr remstg]

;******************************************************************

(defun printsent (remstg)   ; mod 2.4 - called by semanalyzer - prints top
                            ;                  choice sentence in sentstlst out to
                            ;                  user's terminal
        (prog ()
                (setq tempshortermem '())
                (terpr)(terpr)(terpr)
                (princ '|*************************************************|)
                (terpr)(terpr)(princ '|Sperexsys output to user:|)(terpr)
                (outrestsent remstg)]

;******************************************************************
;******************************************************************


;******************************************************************
;******************************************************************

(defun userfdbk ()   ; mod 2.5 - called by semanalyzer - asks user for yes/no
                     ;                  feedback on correctness of sentence just
                     ;                  printed.
        (prog (check)
                loop
                (terpr)(terpr)(terpr)
            (princ '|Is the above sentence correct? (Type "(yes)" or "(no)")|)
            (terpr)(princ '|> |)
            (setq check (read 'check))
            (cond
                  ((equal check '(yes))(return t))
```

```
                ((equal check '(no)) (return nil)))
            (go loop]

;**********************************************************************
;**********************************************************************


;**********************************************************************
;**********************************************************************

(defun reinit (check)   ; mod 2.6 - called by semanalyzer - resets the error
                        ;             parameters based on epfe's past performance
   (prog ()
      (cond
         (init t)
         (t (cond
              (check         ; have found correct sentence
                    (setq shortermem (append tempshortermem shortermem))
                             ; here's where the temporary short term memory
                             ; is added to the real short term memory
                    (setq inittim (gettim1 (cdr sentout)))
                    (setq acceptthresh (add .01 acceptthresh))
                    (cond
                        ((greaterp searchdepth 2)
                             (setq searchdepth (diff searchdepth 1)))))
              (t
                (terpr)(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)
                (princ '|**********************************************************************|)
                (terpr)(terpr)
                (princ '|    I'm sorry, but the SPEREXSYS has failed to properly interpret this last |)(terpr)
                (princ '|    Please repeat the sentence giving particular care to the pronunciation|)(terpr)
                (princ '|    of the words which were improperly identified.|)(terpr)(terpr)
                (princ '|    Hit the return key after you have done so.|)
                (terpr)(terpr)
                (princ '|**********************************************************************|)
                (terpr)(terpr)(terpr)(terpr)(terpr)(terpr)(terpr)
                (princ '|> |)
                ; If a real voice decoder were being used, it would be
                ; reset at this point.
                (readc)(readc)
                (setq acceptthresh (diff acceptthresh .05))
                (setq searchdepth (add searchdepth 2))))))
      (terpr)(princ '|from reinit: acceptthresh = |)(print acceptthresh)
      (princ '|and searchdepth = |)(print searchdepth)(princ '|)|)]


;**********************************************************************
;**********************************************************************


;**********************************************************************
;**********************************************************************

(defun semanalyzer ()   ; mod 2 - called by sperexsys
      (prog ()
            (semaninit)
            loop1
            (epfe searchdepth sentstart acceptthresh inittim)
            (setq sentstlst (ranksents))
            loop2
            (cond
```

```
                        ((null sentstlst)(reinit nil))
                        (t
                             (setq sentout (cdar sentstlst))
                                                    ;cdar sentstlst =  words (no string number)
                                                    ; of sentence
(terpr)(princ '|from mod 2: sentout = |)(print sentout)(terpr)(princ '|and sentstlst = |)(print sentstlst)(print)
                             (printsent sentout)
                             (cond
                                 ((userfdbk)(reinit t))
                                 (t (setq sentstlst (cdr sentstlst))(go loop2)))))
                    (go loop1)]

;*************************************************************************
;*************************************************************************
;*************************************************************************
;*************************************************************************


;*************************************************************************
;*************************************************************************
;*************************************************************************
;*************************************************************************

(defun shortermemprob (word prob)    ; mod 3.1 - called by shorttermmem -
                                     ;            increases word prob if word is
                                     ;            in short term memory
         ; new prob (if in short term memory) = prob +  (1 - prob)/3
         (cond
                  ((member word shortermem)(add .33 (times .67 prob)))
                  (t prob)]

;*************************************************************************

(defun shorttermmem (words)  ; mod 3 - referenced by semantic analyzer and epfe
                             ;      - inputs to epfe through procwdgs -
                               ;          modifies the probabilities  of words in
                               ;          wdgs based on whether or not they have
                               ;          recently been spoken in a user approved
                               ;          sentence.  Returns the modified wdgs.
         (cond
                  ((equal shortermem '()) words)
                  ((null words) '())
                  (t
                   (setq word (car words))
                   (setq prob (caddr word))
                   (setq times (cadr word))
                   (setq worddict (car word))
                   (setq prob (shortermemprob worddict prob))
                   (setq word (cons worddict (cons times (list prob))))
                   (cons word (shorttermmem (cdr words)]

;*************************************************************************
;*************************************************************************
;*************************************************************************
;*************************************************************************
```

```
;*************************** SPEREXSYS ********************************

; SPEREXSYS - top level - functions as a driver for the system.
(setsyntax '#    'splicing 'toor)
(defmacro toor () (prog () (return (list 'or))))
(setsyntax ' 'splicing 'toand)
(defmacro toand () (prog () (return (list 'and))))


(spxsinit)
(semanalyzer)

;*************************************************************************
;
;
;    THIS IS THE VOCABULARY/DICTIONARY FOR THE VOICE DECODER WHICH
;                IS USED BY THE SPEREXSYS
;
;
;*************************************************************************


(defun vocdict ()
   (prog ()
          (setq all '(the to that tee tea air airforce error err or force farce
fierce fear system general gendre gent gents cent cents scent gem gym nor
 all awl roll was wash want wall a as speaking peaking speak peak peek peeking
 speech pacing peachy king tow twist his hiss staff stay aft after fun about
 bout out abort some sum sun such summer more recent recess regency regent
 c3 see sea cubed cuba cue bed dish dishes issues itches itch you ewes she
 he said told door dare there their they wrist risk is snow no know noting
nothing naughty thin think thing gamble am big us ambiguous pick ambient
 sub this dizzy interest enter rest sin center central intelligence intelligent
 alliegence sensor repeat report port reap army arm me our are medium median
 mi my eye might be people peep hole pole poll land and an row own round
 dinn inn in intel into telephone folks foes foe foal vote owl agree green
 enemy enema run running short shout ton on down ammunition we have got
 goat communist communications communicator get kit ghetto inform information
 uniform units eunichs one two three four five six seven eight nine zero
 sentence word number right wrong eoeoeo has))

          (setq fpunct all)
          (setq fpunc all)
          (setq noun '(tee tea air force system general gem awl wall peak speech
 king staff sum sun summer recess regency c3 see sea dish itch snow gamble
 pick ambient interest rest center report port arm median eye people hole
 pole poll land telephone foal enema goat kit ghetto information uniform
 sentence word squadron fear roll wash speak peek tow twist hiss stay bout
 abort see cue dare risk know think enter sin sensor repeat reap row own
 vote run shout get inform gents dishes issues itches ewes folks foes communications
 units eunichs error farce gendre gent cent scent gym aft regent cuba there
 nothing sub intelligence alliegence mi dinn inn foe owl enemy ammunition
 communicator cents wrist airforce door peep intel))
          (setq n1p '(this me our my we))
          (setq det '(the that a this an))
          (setq wh '())
          (setq tnsless '(tee tea air force system general gem awl wall peak speech
 king staff sum sun summer recess regency c3 see sea dish itch snow gamble
 pick ambient interest rest center report port arm median eye people hole
 pole poll land telephone foal enema goat kit ghetto information uniform
 sentence word squadron fear roll wash speak peek tow twist hiss stay bout
 abort see cue dare risk know think enter sin sensor repeat reap row own
 vote run shout get inform err want wrist airforce door no army be peep
```

```
intel agree have see know))
        (setq future '(might))
        (setq en '(cubed said told no noting got))
        (setq verb '(tee tea air force system general gem awl wall peak speech
king staff sum sun summer recess regency c3 see sea dish itch snow gamble
pick ambient interest rest center report port arm median eye people hole
pole poll land telephone foal enema goat kit ghetto information uniform
sentence word squadron fear roll wash speak peek tow twist hiss stay bout
abort see cue dare risk know think enter sin sensor repeat reap row own
vote run shout get inform to err was want speaking peaking peeking pacing
cubed said told wrist airforce door is am army are be peep intel agree
have got see know has))
        (setq v1s '(am))
        (setq vspl '(c3 cubed said told got))
        (setq adj '(general fierce peachy recent naughty thin big ambiguous dizzy
central intelligent medium round green short communist right wrong speaking
peaking peeking pacing fun more mi noting))
        (setq relpron '())
        (setq comma '())
        (setq have '(have has))
        (setq do '())
        (setq for '())
        (setq inf_comp '(that want said told have got see know has))
        (setq to_less_inf_comp '(have see has))
        (setq two_obj '(told))
        (setq name '())
        (setq variable '(a us))
        (setq than '())
        (setq quantifier '(all some))
        (setq be_'(be))
        (setq comp '())
        (setq of '())
        (setq ngstart '(tee tea air force system general gem awl wall peak speech
king staff sum sun summer recess regency c3 see sea dish itch snow gamble
pick ambient interest rest center report port arm median eye people hole
pole poll land telephone foal enema goat kit ghetto information uniform
sentence word squadron gents dishes issues itches ewes folks foes communications
units eunichs error farce gendre gent cent scent gym aft regent cuba there
nothing sub intelligence alliegence mi dinn inn foe owl enemy ammunition
communicator one two three four five six seven eight nine zero fierce peachy
recent naughty thin big ambiguous dizzy central intelligent medium round
green short communist right wrong the that cents all a peaking peeking
some more cubed dishes itches issues you she he their they us this army
my an intel we wrist airforce door me our no))
        (setq n2p '(you))
        (setq def '(the this))
        (setq ns '(tee tea air force system general gem awl wall peak speech king
staff sum sun summer recess regency c3 see sea dish itch snow gamble pick
ambient interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word squadron fear roll wash speak peek tow twist hiss stay bout abort
see cue dare risk know think enter sin sensor repeat reap row own vote
run shout get inform error farce gendre gent cent scent gym aft regent
cuba there nothing sub intelligence alliegence mi dinn inn foe owl enemy
ammunition communicator one two three four five six seven eight nine zero
the that a speaking peaking peeking pacing more cubed you she he their
this army arm me our my peep an intel ton wrist airforce door))
        (setq past '(was cubed said told got))
        (setq modal '(might))
        (setq ing '(speaking peaking peeking pacing noting))
        (setq auxverb '(was is am are be have has))
```

```
        (setq v3s '(squadron gents dishes issues itches ewes folks foes
communications units eunichs cents speaking peaking peeking is us army has))
        (setq vpl_2s '(are))
        (setq prep '(to as after about out in into on down))
        (setq ord '())
        (setq poss '(their my our))
        (setq be '(was is am are be))
        (setq conj '(or nor and))
        (setq that_comp '(said told know))
        (setq to_be_less_inf_comp '())
        (setq propnoun '(mi cuba intel there))
        (setq that '(that))
        (setq sent_subj '(is))
        (setq unit '(cents cent ton))
        (setq n3p '(tee tea air force system general gem awl wall peak speech king
staff sum sun summer recess regency c3 see sea dish itch snow gamble pick
ambient interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word squadron fear roll wash speak peek tow twist hiss stay bout abort
see cue dare risk know think enter sin sensor repeat reap row own vote
run shout get inform gents dishes issues itches ewes folks foes communications
units eunichs error farce gendre gent cent scent gym aft regent cuba there
nothing sub intelligence alliegence mi dinn inn foe owl enemy ammunition
communicator the cents a speaking peeking peaking pacing more cubed she
he their they this army peep an intel wrist airforce door))
        (setq indef '(all a an))
        (setq npl '(gents dishes issues itches ewes folks foes communications units
eunichs the cents all some you they us our my we ammunition))
        (setq pres '(tee tea air force system general gem awl wall peak speech
king staff sum sun summer recess regency c3 see sea dish itch snow gamble
pick ambient interest rest center report port arm median eye people hole
pole poll land telephone foal enema goat kit ghetto information uniform
sentence word squadron fear roll wash speak peek tow twist hiss stay bout
abort see cue dare risk know think enter sin sensor repeat reap row own
vote run shout get inform gents dishes issues itches ewes folks foes communications
units eunichs to err cents want speaking peeking pacing wrist airforce
door is noting am us army are peep intel agree have know has))
        (setq neg '(nor no noting))
        (setq v_3s '(fear roll wash speak peek tow twist hiss stay bout abort see cue dare risk know think et
        (setq v13s '(was))
        (setq pronoun '(that more you she he their they this nothing me our my
                                    we army))
        (setq adverb '(all some more nothing))
        (setq dim '())
        (setq to '(to))
        (setq how '())
        (setq no_subj '(want said told))
        (setq time '())
        (setq quant '(one two three four five six seven eight nine zero us no))
        (setq compadv '(such))]

;****************************************************************************

(vocdict)
(terpr)
(terpr)
(princ '|Vocdict has been loaded and executed.|)(terpr)

;****************************************************************************
```

```
;****************************************************************************
;
;           dictionary for sperexsys
;
;****************************************************************************

(defun initvocab ()
  (prog()
      (setq featureset '(all fpunct fpunc noun n1p det wh tnsless future en
         verb v1s vspl adj relpron comma have do for inf_comp to_less_inf_comp
         two_obj name variable than quantifier be_comp of ngstart n2p def
         ns past modal ing auxverb v3s vpl_2s prep ord poss be conj that_comp
         to_be_less_inf_comp propnoun that sent_subj unit n3p indef npl pres
         neg v_3s v13s pronoun adverb dim to how no_subj time quant compadv))

      (setq notfeatset '(qp q p a v np s ap trace than_comp poss_np andc passive
     wh_quest
         np_utterance inf prog predp part copula perf wh_comp pp_utterance
         ynquest decl imperative relpron_np comp_s relative possesive
         gp major pp aux vp binder sec))]

;****************************************************************************

(initvocab)
(terpr)
(princ '|Dict.spxs has been loaded and executed.|)(terpr)
```

APPENDIX B

A SAMPLE RUN OF THE SPEREXSYS

B. A Sample Run of the SPEREXSYS

This is a sample SPEREXSYS run of the recognition of the sentence: "The peak got snow." Comments are included on the listing to assist the reader in understanding it. An analysis of this run, and a discussion of what it demonstrates about the performance of the SPEREXSYS, is presented in the "Test Results and Conclusions" portion of the "Test Number One" section in chapter IV.

An explanation of how the simulated input from the Voice Decoder was chosen is described in chapter IV.

{ NOTE TO THESIS READER:

THIS LISTING IS THE ACTUAL SCRIPT LISTING OF THE RUN FOR TEST NUMBER ONE. IT HAS BEEN EDITED IN THAT:

1. ALL VERTICAL BARS HAVE BEEN REMOVED FROM THE FILE WHICH WERE INSERTED BY THE LISP SHELL DURING THE THE RECEPTION OF INFORMATION FROM THE DEC-10 (THE ENGLISH PARSER). THIS WAS DONE ONLY TO MAKE THE LISTING MORE READABLE.

2. CARRIAGE RETURNS HAVE BEEN INSERTED EVERY 80 CHARACTERS IN THOSE LINES WHICH EXCEEDED 80 CHARACTERS. THIS WAS NECESSARY BECAUSE THE PRINTER WHICH PRINTED THIS FILE DOES NOT HAVE A WRAPAROUND FEATURE AND THE LETTERS BEYOND THE 80'S WERE GETTING OVERPRINTED AT THE END OF THE LINE.

3. COMMENTS HAVE BEEN ADDED TO THE SCRIPT AFTER THE RUN IN ORDER TO ASSIST THE READER IN UNDERSTANDING THE RUN. THEY ARE ALL IN ALL CAPITAL LETTERS AND ARE ENCLOSED IN BRACES SUCH AS THIS ONE. }

```
Script started on Wed Jul 13 18:06:43 1983
Warning: no access to tty; thus no job control in this shell...
% lisp
Franz Lisp, Opus 36
-> (load 'spxs)
```

{ THE USER UTTERED (SIMULATED) SENTENCE WAS:

"THE PEAK GOT SNOW."

REFER TO CHAPTER FOUR, TEST NUMBER ONE FOR FURTHER DETAILS. }

```
*******************************************************************
***                                                ***
***  Welcome to the SPoken English Recognition EXpert SYStem   ***
***                  (SPEREXSYS)                   ***
***                                                ***
*******************************************************************
```

Please ready the Milne English Parser and the AFIT Acoustic analyzer.
When they have been readied, input the device I.D. of the
English Parser. It should be of the following form: /dev/ttyi7

> /dev/tty12

{ THE PROMPT "> " WILL APPEAR EVERY TIME THE SPEREXSYS REQUESTED INPUT FROM THE USER. }

Control has now been turned over to the semantic analyzer.
This is the highest level of decision making in the speech recognition process.
In order to initialize the system error parameters, please answer the following questions.

How many words deep will the Acoustic Analyzer have to go in order to
guarantee that the correct word will be recognized? (Normally this is "3").
> 2 { SEARCHDEPTH = 2 }

What is the minimun acceptable average probabilty of correctness for the last
three words in a string? (Normally this is .75).
> .75 { ACCEPTTHRESH = .75 }

Voc.dict has been loaded and executed.

Dict.spxs has been loaded and executed.

On exiting formnxgs: nxgslst = ((1000 0 (all)))

Output from epfe to voice decoder follows:
Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****
Possible words for voice decoder to choose from are:
ALL WORDS
****

```
Next-guess-request = (1000 0 (all))
               { | |                 | |  LEGAL GRAMMAR TYPES OF THE NEXT WORD
               |                   |                 |  APPROXIMATE START TIME OF NEXT WORD
                                                     STRING NUMBER }
```

```
> (1000
(the (0 15) .95)
(a (0 15) .84)
(they (0 15) .52))
{    | |       | |  PROBABILITY OF LIKELIHOOD THAT THE WORD "they" WAS THE
     |                       USER'S INTENDED NEXT WORD.
     |        |   TERMINATION TIME FOR THEW WORD IN THE INPUT UTTERANCE.
                    START TIME FOR THE WORD IN THE INPUT UTTERANCE. }
```

This concludes output (*next-guess-requests*) *from the epfe to the voice* decoder.

Before entering dectopwds: wordgslst = ((1000 (the (0 15) 0.95) (a (0 15) 0.84)
(they (0 15) 0.52)))

On exiting dectopwords: topwordlist = ((1000 (the (0 15) 0.951875) (a (0 15) 0.8
46) (they (0 15) 0.538)))

After exiting startnsts: stringlist = ((1003 (they (0 15) 0.538)) (1002 (a (0 1
5) 0.846)) (1001 (the (0 15) 0.951875)))
After exiting killowsts: stringlist = ((1003 (they (0 15) 0.538)) (1002 (a (0 1
5) 0.846)) (1001 (the (0 15) 0.951875)))

To summarize the above stringlist, the following strings are still active:

```
they  { ALL THREE INPUT WORDS SURVIVED (EVEN THOUGH SEARCHDEPTH = 2)
a       ONLY BECAUSE FOR THE FIRST WORD IN A SENTENCE, SEARCHDEPTH SQUARED
the     (IN THIS CASE 4) SURVIVORS ARE ALLOWED. }
```

Data from epfe to english parser follows:
go1([they, pause]).

~ { LINE HIT CAUSES ACTUAL DEC-10 OUTPUT TO BE IGNORED.
THIS WILL NECESSITATE RETRYING THE DATA EXCHANGE
BETWEEN THE VAX AND THE DEC-10 AT THE END OF THIS ONE. }

```
Data from epfe to english parser follows:
go1([a,pause]).


[[conj]       {
              {
[possesive]   {
              {
[conj]        {
              {  THIS IS ALL BEING IGNORED DUE TO THE LINE HIT
[verb]        {    DESCRIBED ABOVE. }
              {
[possesive]   {
              {
]             {

yes

((conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (quant)
(noun and not npl) (noun and npl) (noun and not npl) (than) (quant))



Data from epfe to english parser follows:
go1([the,pause]).


yes

((conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (quant)
(noun and not npl) (noun and npl) (noun and not npl) (than) (quant))


(epreslst is as follows:)
((1001 (conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (qua
nt) (noun and not npl) (noun and npl) (noun and not npl) (than) (quant)) (1002
(conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (quant) (no
un and not npl) (noun and npl) (noun and not npl) (than) (quant)) (1003 . ¬))

Would you like to try the EP interface again?
 (r - rerun; i - new instrs; g - keep going)
>r  { BECAUSE OF LINE HIT ABOVE, WE NEED TO RERUN THE ENTIRE DATA EXCHANGE. }


Data from epfe to english parser follows:
go1([they,pause]).

yes

((conj) (possesive) (conj) (verb) (possesive)) { THIS TIME IT GOT IT. }



Data from epfe to english parser follows:
go1([a,pause]).


yes
```

((conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (quant) (noun and not npl) (noun and npl) (noun and not npl) (than) (quant))

Data from epfe to english parser follows:
go1([the, pause]).

yes

((conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (quant) (noun and not npl) (noun and npl) (noun and not npl) (than) (quant))

(epreslst is as follows:)
((1001 (conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (qua
nt) (noun and not npl) (noun and npl) (noun and not npl) (than) (quant)) (1002
(conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than) (quant) (no
un and not npl) (noun and npl) (noun and not npl) (than) (quant)) (1003 (conj)
(possesive) (conj) (verb) (possesive)))

Would you like to try the EP interface again?
(r - rerun; i - new instrs; g - keep going)
>g { THIS TIME THE ENTIRE DATA EXCHANGE WENT WELL, SO THE RUN WILL CONTINUE. }

On exiting formnxgs: nxgslst = ((1003 15 (conj) (possesive) (conj) (verb) (poss
esive)) (1002 15 (conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (
than) (quant) (noun and not npl) (noun and npl) (noun and not npl) (than) (quan
t)) (1001 15 (conj) (quant) (ord) (than) (quant) (adj) (adj and not noun) (than
) (quant) (noun and not npl) (noun and npl) (noun and not npl) (than) (quant)))

Output from epfe to voice decoder follows:
Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****
Possible words for voice decoder to choose from are:

or nor and tee tea air force system general gem awl
wall peak speech king staff sum sun summer recess regency c3
see sea dish itch snow gamble pick ambient interest rest center
report port arm median eye people hole pole poll land telephone
foal enema goat kit ghetto information uniform sentence word squadron fear
roll wash speak peek tow twist hiss stay bout abort see
cue dare risk know think enter sin sensor repeat reap row
own vote run shout get inform to err was want speaking
peaking peeking pacing cubed said told wrist airforce door is am
army are be peep intel agree have got see know has

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 110

{ NOTE - FOR THIS SET OF NEXT-WORD-LEGAL-FEATURES, THE VOCABULARY WHICH THE VOICE DECODER WILL HAVE TO CONSIDER IS REDUCED BY ABOUT HALF. }

****

Next-guess-request = (1003 15 (conj) (possesive) (conj) (verb) (possesive))
> (1003
(peak (15 35) .95) { "Peak" and "peek" ARE ACOUSTICALLY INDISTINGUISHABLE,
(peek (15 35) .95) HENCE, THEY BOTH HAVE IDENTICAL PROBABILITIES OF .95. }
(repeat (5 35) .73))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

or nor and fierce peachy recent naughty thin big ambiguous dizzy
central intelligent medium round green short communist right wrong speaking peaking
peeking pacing fun more noting gents dishes issues itches ewes folks
foes communications units eunichs ammunition cents tee tea air force system
general gem awl wall peak speech king staff sum sun summer
recess regency c3 see sea dish itch snow gamble pick ambient
interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word fear roll wash speak peek tow twist hiss stay bout
abort see cue dare risk know think enter sin sensor repeat
reap row own vote run shout get inform error farce gendre
gent cent scent gym aft regent cuba there nothing sub intelligence
alliegence mi dinn inn foe owl enemy communicator wrist airforce door
peep intel one two three four five six seven eight nine
zero us

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 156

{ THIS TIME THE VOCABULARY CHOICES WERE ONLY REDUCED BY ABOUT 1/4. }

****

Next-guess-request = (1002 15 (conj) (quant) (ord) (than) (quant) (adj) (adj an
d not noun) (than) (quant) (noun and not npl) (noun and npl) (noun and not npl)
(than) (quant))
> (1002
(peak (15 35) .95)
(peek (15 35) .95)
(repeat (5 35) .73))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

or nor and fierce peachy recent naughty thin big ambiguous dizzy
central intelligent medium round green short communist right wrong speaking peaking
peeking pacing fun more noting gents dishes issues itches ewes folks
foes communications units eunichs ammunition cents tee tea air force system
general gem awl wall peak speech king staff sum sun summer
recess regency c3 see sea dish itch snow gamble pick ambient
interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word fear roll wash speak peek tow twist hiss stay bout
abort see cue dare risk know think enter sin sensor repeat
reap row own vote run shout get inform error farce gendre
gent cent scent gym aft regent cuba there nothing sub intelligence

alliegence mi dinn inn foe owl enemy communicator wrist airforce door
peep intel one two three four five six seven eight nine
zero us

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 156
****

Next-guess-request = (1001 15 (conj) (quant) (ord) (than) (quant) (adj) (adj an
d not noun) (than) (quant) (noun and not npl) (noun and npl) (noun and not npl)
(than) (quant))
> (1001
(peak (15 35) .95)
(peek (15 35) .95)
(repeat (5 35) .73))

This concludes output (next-guess-requests) from the epfe to the voice decoder.

Before entering dectopwds: wordgslst = ((1001 (peak (15 35) 0.95) (peek (15 35)
0.95) (repeat (5 35) 0.73)) (1002 (peak (15 35) 0.95) (peek (15 35) 0.95) (rep
eat (5 35) 0.73)) (1003 (peak (15 35) 0.95) (peek (15 35) 0.95) (repeat (5 35)
0.73)))

On exiting dectopwords: topwordlst = ((1003 (peek (15 35) 0.9525) (peak (15 35)
0.9525)) (1002 (peek (15 35) 0.9525) (peak (15 35) 0.9525)) (1001 (peek (15 35
) 0.9525) (peak (15 35) 0.9525)))

After exiting startnsts: stringlist = ((1009 (the (0 15) 0.951875) (peak (15 35
) 0.9525)) (1008 (the (0 15) 0.951875) (peek (15 35) 0.9525)) (1007 (a (0 15) 0
5 (they (0 15) 0.538) (peak (15 35) 0.9525)) (1004 (they (0 15) 0.538) (peek (1
5 35) 0.9525)))
After exiting killowsts: stringlist = ((1009 (the (0 15) 0.951875) (peak (15 35
) 0.9525)) (1008 (the (0 15) 0.951875) (peek (15 35) 0.9525)) (1007 (a (0 15) 0

To summarize the above stringlist, the following strings are still active:

the peak { BECAUSE ONLY FOUR STRINGS ARE ALLOWED TO SURVIVE, ALL STRINGS
the peek BEGINNING WITH "they" (THE LOWEST PROBABILITY THIRD WORD BACK)
a peak HAVE BEEN KILLED. ALSO, "repeat" WAS ELIMINATED IN THE
a peek DECTOPWDS MODULE (SEE TOPWDLST ABOVE). }

Data from epfe to english parser follows:
go1([the, peak, pause]).

yes

((noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that
) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (o
f) (conj) (verb) (possesive))

Data from epfe to english parser follows:
go1([the, peek, pause]).

yes

((noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that
) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (o

f) (conj) (verb) (possesive))

Data from epfe to english parser follows:
go1([a, peak, pause]).

yes

((noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that
) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (o
f) (conj) (verb) (possesive))

Data from epfe to english parser follows:
go1([a, peek, pause]).

yes

((noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that
) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (o
f) (conj) (verb) (possesive))

(epreslst is as follows:)
((1006 (noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (
that) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that
) (of) (conj) (verb) (possesive)) (1007 (noun) (prep) (verb and ing) (verb and
en) (relative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma)
(than) (quant) (det and not that) (of) (conj) (verb) (possesive)) (1008 (noun)
(prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that) (relpro
n_np) (conj and not andc) (comma) (than) (quant) (det and not that) (of) (conj)
(verb) (possesive)) (1009 (noun) (prep) (verb and ing) (verb and en) (relative
) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (quan
t) (det and not that) (of) (conj) (verb) (possesive)))

Would you like to try the EP interface again?
(r - rerun; i - new instrs; g - keep going)
> g

On exiting formnxgs: nxgslst = ((1009 35 (noun) (prep) (verb and ing) (verb and
en) (relative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma
) (than) (quant) (det and not that) (of) (conj) (verb) (possesive)) (1008 35 (n
oun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that) (re
lpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (of) (c
onj) (verb) (possesive)) (1007 35 (noun) (prep) (verb and ing) (verb and en) (r
elative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than
) (quant) (det and not that) (of) (conj) (verb) (possesive)) (1006 35 (noun) (p
rep) (verb and ing) (verb and en) (relative) (relpron and wh) (that) (relpron_n
p) (conj and not andc) (comma) (than) (quant) (det and not that) (of) (conj) (v
erb) (possesive)))

Output from epfe to voice decoder follows:
Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:

(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

gents dishes issues itches ewes folks foes communications units eunichs error
farce gendre gent cent scent gym aft regent cuba there nothing
sub intelligence alliegence mi dinn inn foe owl enemy ammunition communicator
cents as after about out in into on down that one
two three four five six seven eight nine zero us the
a this an or nor and tee tea air force system
general gem awl wall peak speech king staff sum sun summer
recess regency c3 see sea dish itch snow gamble pick ambient
interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word squadron fear roll wash speak peek tow twist hiss stay
bout abort see cue dare risk know think enter sin sensor
repeat reap row own vote run shout get inform to err
was want speaking peaking peeking pacing cubed said told wrist airforce
door is am army are be peep intel agree have got
see know has

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 168

****

Next-guess-request = (1009 35 (noun) (prep) (verb and ing) (verb and en) (relat
ive) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (q
uant) (det and not that) (of) (conj) (verb) (possesive))

> (1009
(out (35 50) .90) { "Out" and "got" ARE BEING INPUT WITH THE SAME PROBABILITIES
(on (35 50) .75) TO SIMULATE THE CONDITION WHEN THE VOICE DECODER IS UNABLE
(got (30 50) .90)) TO FAVOR ONE OF THEM OVER THE OTHER. }

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

gents dishes issues itches ewes folks foes communications units eunichs error
farce gendre gent cent scent gym aft regent cuba there nothing
sub intelligence alliegence mi dinn inn foe owl enemy ammunition communicator
cents as after about out in into on down that one
two three four five six seven eight nine zero us the
a this an or nor and tee tea air force system
general gem awl wall peak speech king staff sum sun summer
recess regency c3 see sea dish itch snow gamble pick ambient
interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word squadron fear roll wash speak peek tow twist hiss stay
bout abort see cue dare risk know think enter sin sensor
repeat reap row own vote run shout get inform to err
was want speaking peaking peeking pacing cubed said told wrist airforce
door is am army are be peep intel agree have got
see know has

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 168

****

Next-guess-request = (1008 35 (noun) (prep) (verb and ing) (verb and en) (relat
ive) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (q
uant) (det and not that) (of) (conj) (verb) (possesive))

```
> (1008
(out (35 50) .90)
(on (35 50) .75)
(got (30 50) .90))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

**************************************************************************
Possible words for voice decoder to choose from are:

 gents dishes issues itches ewes folks foes communications units eunichs error
 farce gendre gent cent scent gym aft regent cuba there nothing
 sub intelligence alliegence mi dinn inn foe owl enemy ammunition communicator
 cents as after about out in into on down that one
 two three four five six seven eight nine zero us the
 a this an or nor and tee tea air force system
 general gem awl wall peak speech king staff sum sun summer
 recess regency c3 see sea dish itch snow gamble pick ambient
 interest rest center report port arm median eye people hole pole
 poll land telephone foal enema goat kit ghetto information uniform sentence
 word squadron fear roll wash speak peek tow twist hiss stay
 bout abort see cue dare risk know think enter sin sensor
 repeat reap row own vote run shout get inform to err
 was want speaking peaking peeking pacing cubed said told wrist airforce
 door is am army are be peep intel agree have got
 see know has

   TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 168
**************************************************************************
Next-guess-request = (1007 35 (noun) (prep) (verb and ing) (verb and en) (relat
ive) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (q
uant) (det and not that) (of) (conj) (verb) (possesive))
> (1007
(out (35 50) .9)
(on (35 50) .75)
(got (30 50) .90))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

**************************************************************************
Possible words for voice decoder to choose from are:

 gents dishes issues itches ewes folks foes communications units eunichs error
 farce gendre gent cent scent gym aft regent cuba there nothing
 sub intelligence alliegence mi dinn inn foe owl enemy ammunition communicator
 cents as after about out in into on down that one
 two three four five six seven eight nine zero us the
 a this an or nor and tee tea air force system
 general gem awl wall peak speech king staff sum sun summer
 recess regency c3 see sea dish itch snow gamble pick ambient
 interest rest center report port arm median eye people hole pole
 poll land telephone foal enema goat kit ghetto information uniform sentence
 word squadron fear roll wash speak peek tow twist hiss stay
 bout abort see cue dare risk know think enter sin sensor
 repeat reap row own vote run shout get inform to err
 was want speaking peaking peeking pacing cubed said told wrist airforce
 door is am army are be peep intel agree have got
```

see know has

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 168
****

Next-guess-request = (1006 35 (noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (of) (conj) (verb) (possesive))
> (1006
(out (35 50) .90)
(on (35 50) .75)
(got (30 50) .90))

This concludes output (next-guess-requests) from the epfe to the voice decoder.

Before entering dectopwds: wordgslst = ((1006 (out (35 50) 0.9) (on (35 50) 0.75) (got (30 50) 0.9)) (1007 (out (35 50) 0.9) (on (35 50) 0.75) (got (30 50) 0.9)) (1008 (out (35 50) 0.9) (on (35 50) 0.75) (got (30 50) 0.9)) (1009 (out (35 50) 0.9) (on (35 50) 0.75) (got (30 50) 0.9)))

On exiting dectopwords: topwordlst = ((1009 (got (30 50) 0.905) (out (35 50) 0.90375)) (1008 (got (30 50) 0.905) (out (35 50) 0.90375)) (1007 (got (30 50) 0.905) (out (35 50) 0.90375)) (1006 (got (30 50) 0.905) (out (35 50) 0.90375)))

After exiting startnsts: stringlist = ((1017 (a (0 15) 0.846) (peek (15 35) 0.9525) (out (35 50) 0.90375)) (1016 (a (0 15) 0.846) (peek (15 35) 0.9525) (got (30 50) 0.905)) (1015 (a (0 15) 0.846) (peak (15 35) 0.9525) (out (35 50) 0.90375)) (1014 (a (0 15) 0.846) (peak (15 35) 0.9525) (got (30 50) 0.905)) (1013 (the (0 15) 0.951875) (peek (15 35) 0.9525) (out (35 50) 0.90375)) (1012 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905)) (1011 (the (0 15) 0.951875) (peak (15 35) 0.9525) (out (35 50) 0.90375)) (1010 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905)))

A decision is now being made on the third word from the end of all strings.
The choices are:
(the a)

After exiting killowsts: stringlist = ((1013 (the (0 15) 0.951875) (peek (15 35) 0.9525) (out (35 50) 0.90375)) (1012 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905)) (1011 (the (0 15) 0.951875) (peak (15 35) 0.9525) (out (35 50) 0.90375)) (1010 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905)))

To summarize the above stringlist, the following strings are still active:

the peek out { THE STRINGS STARTING WITH THE WORD "a" HAVE BEEN KILLED. }
the peek got
the peak out
the peak got

Data from epfe to english parser follows:
go1([the,peek,out,pause]).

yes

((ngstart))

```
Data from epfe to english parser follows:
go1([the,peek,got,pause]).


yes

((conj) (conj) (poss_np) (adverb) (prep) (for and pp) (compadv) (name and not
 np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (co
nj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (fpunc) (s
ent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (det) (
ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (pro
noun))



Data from epfe to english parser follows:
go1([the,peak,out,pause]).


yes

((ngstart))



Data from epfe to english parser follows:
go1([the,peak,got,pause]).


yes

((conj) (conj) (poss_np) (adverb) (prep) (for and pp) (compadv) (name and no
t np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (c
onj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (fpunc) (
sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (det)
(ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (pr
onoun))


(epreslst is as follows:)
((1010 (conj) (conj) (poss_np) (adverb) (prep) (for and pp) (compadv) (name and
 not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp)
 (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (fpunc
) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (de
t) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s)
(pronoun)) (1011 (ngstart)) (1012 (conj) (conj) (poss_np) (adverb) (prep) (for
and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (p
ronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (pr
ep) (conj) (poss_np) (fpunc) (sent_subj) (comma) (comma) (compadv) (name and no
t np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (c
onj and not andc) (comp_s) (pronoun)) (1013 (ngstart)))

Would you like to try the EP interface again?
 (r - rerun; i - new instrs; g - keep going)
>g


On exiting formnxgs: nxgslst = ((1013 50 (ngstart)) (1012 50 (conj) (conj) (pos
s_np) (adverb) (prep) (for and pp) (compadv) (name and not np) (propnoun) (prep
```

) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (com p_s) (adverb) (pronoun) (prep) (conj) (poss_np) (sent_subj) (comma) (comma) (co mpadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or d et)) (than_comp) (conj and not andc) (comp_s) (pronoun)) (1011 50 (ngstart)) (1 010 50 (conj) (conj) (poss_np) (adverb) (prep) (for and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (det) (ngst art and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (pronoun)))

Output from epfe to voice decoder follows:
Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****
Possible words for voice decoder to choose from are:

tee tea air force system general gem awl wall peak speech
king staff sum sun summer recess regency c3 see sea dish
itch snow gamble pick ambient interest rest center report port arm
median eye people hole pole poll land telephone foal enema goat
kit ghetto information uniform sentence word squadron gents dishes issues itches
ewes folks foes communications units eunichs error farce gendre gent cent
scent gym aft regent cuba there nothing sub intelligence alliegence mi
dinn inn foe owl enemy ammunition communicator one two three four
five six seven eight nine zero fierce peachy recent naughty thin
big ambiguous dizzy central intelligent medium round green short communist right
wrong the that cents all a peaking peeking some more cubed
dishes itches issues you she he their they us this army
my an intel we wrist airforce door me our

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 141
****
Next-guess-request = (1013 50 (ngstart))
> (1013
(foe (50 70) .81)
(snow (50 70) .90)
(zero (50 70) .73))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****
Possible words for voice decoder to choose from are:

is such to as after about out in into on down
the a an tee tea air force system general gem awl
wall peak speech king staff sum sun summer recess regency c3
see sea dish itch snow gamble pick ambient interest rest center
report port arm median eye people hole pole poll land telephone
foal enema goat kit ghetto information uniform sentence word gents dishes
issues itches ewes folks foes units eunichs error farce gendre gent
cent scent gym aft regent cuba there sub intelligence alliegence mi
dinn inn foe owl enemy ammunition communicator one two three four
five six seven eight nine zero fierce peachy recent naughty thin
big ambiguous dizzy central intelligent medium round green short communist right
wrong cents all peaking peeking some cubed dishes itches issues us
intel wrist airforce door or nor and that more you she

he their they this nothing me our my we army

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 153

****

```
Next-guess-request = (1012 50 (conj) (conj) (poss_np) (adverb) (prep) (for and
pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (prono
un or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (prep)
(conj) (poss_np) (sent_subj) (comma) (comma) (compadv) (name and not np) (propn
oun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not
andc) (comp_s) (pronoun))
> (1012
(no (55 70) .95)      { "No" IS GOING IN WITH A HIGHER PROBABILITY OF LIKELIHOOD
(snow (50 70) .90)     THAN THE WORD "snow" (WHICH IS THE CORRECT WORD). THIS
(foe (50 70) .81)      THIS IS TO SEE IF THE SPEREXSYS CAN PROPERLY APPLY
(zero (50 70) .73))    SYNTACTIC CONSTRAINTS TO OVERCOME VOICE DECODER
                       INACCURACIES. }
```

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

tee tea air force system general gem awl wall peak speech
king staff sum sun summer recess regency c3 see sea dish
itch snow gamble pick ambient interest rest center report port arm
median eye people hole pole poll land telephone foal enema goat
kit ghetto information uniform sentence word squadron gents dishes issues itches
ewes folks foes communications units eunichs error farce gendre gent cent
scent gym aft regent cuba there nothing sub intelligence alliegence mi
dinn inn foe owl enemy ammunition communicator one two three four
five six seven eight nine zero fierce peachy recent naughty thin
big ambiguous dizzy central intelligent medium round green short communist right
wrong the that cents all a peaking peeking some more cubed
dishes itches issues you she he their they us this army
my an intel we wrist airforce door me our

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 141

****

```
Next-guess-request = (1011 50 (ngstart))
> (1011
(foe (50 70) .81)
(snow (50 70) .90)
(zero (50 70) .73))
```

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

is such to as after about out in into on down
the a an tee tea air force system general gem awl
wall peak speech king staff sum sun summer recess regency c3
see sea dish itch snow gamble pick ambient interest rest center
report port arm median eye people hole pole poll land telephone
foal enema goat kit ghetto information uniform sentence word gents dishes
issues itches ewes folks foes units eunichs error farce gendre gent
cent scent gym aft regent cuba there sub intelligence alliegence mi

dinn inn foe owl enemy ammunition communicator one two three four
five six seven eight nine zero fierce peachy recent naughty thin
big ambiguous dizzy central intelligent medium round green short communist right
wrong cents all peaking peeking some cubed dishes itches issues us
intel wrist airforce door or nor and that more you she
he their they this nothing me our my we army

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 153
****

Next-guess-request = (1010 50 (conj) (conj) (poss_np) (adverb) (prep) (for and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (pronoun))
> (1010
(no (55 70) .95)
(snow (50 70) .90)
(foe (50 70) .81)
(zero (50 70) .73))

This concludes output (next-guess-requests) from the epfe to the voice decoder.

Before entering dectopwds: wordgslst = ((1010 (no (55 70) 0.95) (snow (50 70) 0 70) 0.9) (zero (50 70) 0.73)) (1012 (no (55 70) 0.95) (snow (50 70) 0.9) (foe (50 70) 0.81) (zero (50 70) 0.73)) (1013 (foe (50 70) 0.81) (snow (50 70) 0.9) (zero (50 70) 0.73)))

On exiting dectopwords: topwordlst = ((1013 (snow (50 70) 0.905) (foe (50 70) 0

After exiting startnsts: stringlist = ((1025 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905)) (1024 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905) (no (55 70) 0.951875)) (1023 (the (0 15) 0.951875) (peak (15 35) 0.9525) (out (35 50) 0.90375) (foe (50 70) 0.8195)) (1022 (the (0 15) 0.951875) (peak (15 35) 0.9525) (out (35 50) 0.90375) (snow (50 70) 0.905)) (1021 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905)) (1020 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905) (no (55 70) 0.951875)) (1019 (the (0 15) 0.951875) (peek (15 35) 0.9525) (out (35 50) 0.90375) (foe (50 70) 0.8195)) (1018 (the (0 15) 0. 951875) (peek (15 35) 0.9525) (out (35 50) 0.90375) (snow (50 70) 0.905)))

A decision is now being made on the third word from the end of all strings.
The choices are:
(peek peak)

After exiting killowsts: stringlist = ((1025 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905)) (1024 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905) (no (55 70) 0.951875)) (1021 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905)) (1020 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905) (no (55 7 0) 0.951875)))

To summarize the above stringlist, the following strings are still active:

the peak got snow
the peak got no
the peek got snow
the peek got no

And the following sentences are to be forwarded to the semantic analyzer:

the peek got { SINCE THE ENGLISH PARSER TOLD US THAT THESE WERE COMPLETE
the peak got SENTENCES (AND THEY ARE), THEY WILL BE FORWARDED TO THE
SEMANTIC ANALYZER AS CANDIDATE SENTENCES FROM WHICH THE
SEMANTIC ANALYZER WILL HAVE TO CHOOSE. }

Data from epfe to english parser follows:
go1([the,peak,got,snow,pause]).

yes

((conj) (noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh
) (that) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not t
hat) (of) (conj) (to) (poss_np) (adverb) (prep) (for and pp) (compadv) (name an
d not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp
) (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (fpun
c) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (d
et) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s)
(pronoun))

Data from epfe to english parser follows:
go1([the,peak,got,no,pause]).

yes

((conj) (possesive) (conj) (possesive))

Data from epfe to english parser follows:
go1([the,peek,got,snow,pause]).

yes

((conj) (noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh
) (that) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not t
hat) (of) (conj) (to) (poss_np) (adverb) (prep) (for and pp) (compadv) (name an
d not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp
) (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (fpun
c) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (d
et) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s)
(pronoun))

Data from epfe to english parser follows:
go1([the,peek,got,no,pause]).

yes

((conj) (possesive) (conj) (possesive))

```
(epreslst is as follows:)
((1020 (conj) (possesive) (conj) (possesive)) (1021 (conj) (noun) (prep) (verb
and ing) (verb and en) (relative) (relpron and wh) (that) (relpron_np) (conj an
d not andc) (comma) (than) (quant) (det and not that) (of) (conj) (to) (poss_np
) (adverb) (prep) (for and pp) (compadv) (name and not np) (propnoun) (prep) (d
et) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s)
 (adverb) (pronoun) (prep) (conj) (poss_np) (fpunc) (sent_subj) (comma) (comma)
 (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun
or det)) (than_comp) (conj and not andc) (comp_s) (pronoun)) (1024 (conj) (poss
esive) (conj) (possesive)) (1025 (conj) (noun) (prep) (verb and ing) (verb and
en) (relative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma)
 (than) (quant) (det and not that) (of) (conj) (to) (poss_np) (adverb) (prep) (
for and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and no
t (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun)
 (prep) (conj) (poss_np) (fpunc) (sent_subj) (comma) (comma) (compadv) (name an
d not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp
) (conj and not andc) (comp_s) (pronoun)))

Would you like to try the EP interface again?
 (r - rerun; i - new instrs; g - keep going)
>g


On exiting formnxgs: nxgslst = ((1025 70 (conj) (noun) (prep) (verb and ing) (v
erb and en) (relative) (relpron and wh) (that) (relpron_np) (conj and not andc)
 (comma) (than) (quant) (det and not that) (of) (conj) (to) (poss_np) (adverb)
(prep) (for and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstar
t and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (
pronoun) (prep) (conj) (poss_np) (sent_subj) (comma) (comma) (compadv) (name an
d not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp
) (conj and not andc) (comp_s) (pronoun)) (1024 70 (conj) (possesive) (conj) (p
ossesive)) (1021 70 (conj) (noun) (prep) (verb and ing) (verb and en) (relative
) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (quan
t) (det and not that) (of) (conj) (to) (poss_np) (adverb) (prep) (for and pp) (
compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or
 det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj
) (poss_np) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun)
(prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc)
 (comp_s) (pronoun)) (1020 70 (conj) (possesive) (conj) (possesive)))

Output from epfe to voice decoder follows:
Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

**************************************************************************
Possible words for voice decoder to choose from are:

 squadron fear roll wash speak peek tow twist hiss stay bout
 abort cue dare risk know think enter sin sensor repeat reap
 row own vote run shout get inform communications peep speaking pacing
 said told got is such to as after about out in
 into on down the a an tee tea air force system
 general gem awl wall peak speech king staff sum sun summer
 recess regency c3 see sea dish itch snow gamble pick ambient
 interest rest center report port arm median eye people hole pole
 poll land telephone foal enema goat kit ghetto information uniform sentence
 word gents dishes issues itches ewes folks foes units eunichs error
 farce gendre gent cent scent gym aft regent cuba there sub
 intelligence alliegence mi dinn inn foe owl enemy ammunition communicator one
 two three four five six seven eight nine zero fierce peachy
```

recent naughty thin big ambiguous dizzy central intelligent medium round green
short communist right wrong cents all peaking peeking some cubed dishes
itches issues us intel wrist airforce door or nor and that
more you she he their they this nothing me our my
we army

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 189
****

Next-guess-request = (1025 70 (conj) (noun) (prep) (verb and ing) (verb and en)
(relative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (t
han) (quant) (det and not that) (of) (conj) (to) (poss_np) (adverb) (prep) (for
and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (
pronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (p
rep) (conj) (poss_np) (sent_subj) (comma) (comma) (compadv) (name and not np) (
propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and
not andc) (comp_s) (pronoun))
> (1025
(fpunct (70 100) .90)) { WHEN THE VOICE DECODER SENDS AN "fpunct", THIS
SIGNALS THAT A POSSIBLE SENTENTIAL PAUSE HAS
OCCURRED IN THE INPUT STRING. WHEN ONLY AN
"fpunct" IS SENT (AS IN THIS CASE), IT SIGNALS
THAT THE SPEAKER/USER HAS QUIT SPEAKING (NO
OTHER WORDS FOLLOW). }

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****
Possible words for voice decoder to choose from are:

or nor and

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 3
****

Next-guess-request = (1024 70 (conj) (possesive) (conj) (possesive))
> (1024
(fpunct (70 100) .90))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****
Possible words for voice decoder to choose from are:

squadron fear roll wash speak peek tow twist hiss stay bout
abort cue dare risk know think enter sin sensor repeat reap
row own vote run shout get inform communications peep speaking pacing
said told got is such to as after about out in
into on down the a an tee tea air force system
general gem awl wall peak speech king staff sum sun summer
recess regency c3 see sea dish itch snow gamble pick ambient
interest rest center report port arm median eye people hole pole
poll land telephone foal enema goat kit ghetto information uniform sentence
word gents dishes issues itches ewes folks foes units eunichs error
farce gendre gent cent scent gym aft regent cuba there sub
intelligence alliegence mi dinn inn foe owl enemy ammunition communicator one
two three four five six seven eight nine zero fierce peachy
recent naughty thin big ambiguous dizzy central intelligent medium round green
short communist right wrong cents all peaking peeking some cubed dishes

itches issues us intel wrist airforce door or nor and that
more you she he their they this nothing me our my
we army

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 189
****

Next-guess-request = (1021 70 (conj) (noun) (prep) (verb and ing) (verb and en) (relative) (relpron and wh) (that) (relpron_np) (conj and not andc) (comma) (than) (quant) (det and not that) (of) (conj) (to) (poss_np) (adverb) (prep) (for and pp) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (adverb) (pronoun) (prep) (conj) (poss_np) (sent_subj) (comma) (comma) (compadv) (name and not np) (propnoun) (prep) (det) (ngstart and not (pronoun or det)) (than_comp) (conj and not andc) (comp_s) (pronoun))
> (1021
(fpunct (70 100) .90))

Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:

or nor and

TOTAL NUMBER OF WORDS HAS BEEN REDUCED FOR THIS OPTION FROM 200 TO 3
****

Next-guess-request = (1020 70 (conj) (possesive) (conj) (possesive))
> (1020
(fpunct (70 100) .90))

This concludes output (next-guess-requests) from the epfe to the voice decoder.

Before entering dectopwds: wordgslst = ((1020 (fpunct (70 100) 0.9)) (1021 (fpunct (70 100) 0.9)) (1024 (fpunct (70 100) 0.9)) (1025 (fpunct (70 100) 0.9)))

On exiting dectopwords: topwordlst = ((1025 (fpunct (70 100) 0.9075)) (1024 (fpunct (70 100) 0.9075)) (1021 (fpunct (70 100) 0.9075)) (1020 (fpunct (70 100) 0

After exiting startnsts: stringlist = nil

{ ** }
{ ** }

Epfe done. Returning to semantic analyzer.

{ ** }
{ ** }

(from mod 2: sentout = ((the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905) (fpunct (70 100) 100.9))
and sentstlst = ((1025 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905) (fpunct (70 100) 100.9)) (1021 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905) (snow (50 70) 0.905) (fpunct (70 100) 100.9)) (1010 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50) 0.905)) (1012 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905))))

****

Sperexsys output to user:
the peak got snow. { THIS IS THE SPEREXSYS BEST GUESS AS TO THE IDENTITY OF THE UTTERED SENTENCE. IT IS CORRECT. }

Is the above sentence correct? (Type "(yes)" or "(no)")
> (no) { THE ANSWER TO THIS QUESTION SHOULD BE "(yes)". THE ANSWER "(no)" WAS ENTERED IN ORDER TO DEMONSTRATE THE ABILITY OF THE SPEREXSYS TO OFFER MORE THAN ONE PROBABLE SENTENCE IDENTITY AND THEN TO RECOVER IF NONE OF THESE ARE CORRECT. }

(from mod 2: sentout = ((the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50
) 0.905) (snow (50 70) 0.905) (fpunct (70 100) 100.9))
and sentstlst = ((1021 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50
) 0.905) (snow (50 70) 0.905) (fpunct (70 100) 100.9)) (1010 (the (0 15) 0.9518
75) (peak (15 35) 0.9525) (got (30 50) 0.905)) (1012 (the (0 15) 0.951875) (pee
k (15 35) 0.9525) (got (30 50) 0.905))))

****

Sperexsys output to user:
the peek got snow.

Is the above sentence correct? (Type "(yes)" or "(no)")
> (no)

(from mod 2: sentout = ((the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50
) 0.905))
and sentstlst = ((1010 (the (0 15) 0.951875) (peak (15 35) 0.9525) (got (30 50
) 0.905)) (1012 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50) 0.905)
)))

****

Sperexsys output to user:
the peak got.

Is the above sentence correct? (Type "(yes)" or "(no)")
> (no)

(from mod 2: sentout = ((the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50
) 0.905))
and sentstlst = ((1012 (the (0 15) 0.951875) (peek (15 35) 0.9525) (got (30 50
) 0.905))))

****

Sperexsys output to user:
the peek got.

Is the above sentence correct? (Type "(yes)" or "(no)")
> (no)

****

I'm sorry, but the SPEREXSYS has failed to properly interpret this last sentence.

Please repeat the sentence giving particular care to the pronunciation of the words which were improperly identified.

Hit the return key after you have done so.

****

>

(from reinit: acceptthresh = 0.7 and searchdepth = 4) { ERROR MARGINS ARE EXPANDED TO GIVE THE SYSTEM A BETTER CHANCE TO FIND THE RIGHT SENTENCE THE ON THE NEXT TRY. }

Vocdict has been loaded and executed.

Dict.spxs has been loaded and executed.

On exiting formnxgs: nxgslst = ((1000 0 (all)))

Output from epfe to voice decoder follows:
Please type in the voice decoder's response to the following next-guess-request.
Remember to use the following format:
(stringnum (dict.name1 (tim1 tim2) prob)(dict.name2 (tim1 tim2) prob)...)

****

Possible words for voice decoder to choose from are:
ALL WORDS

****

Next-guess-request = (1000 0 (all))

>

{ AND IT ALL STARTS ALL OVER AGAIN. }

APPENDIX C

USER'S MANUAL FOR THE SPERXSYS

C. User's Manual for the SPEREXSYS

The following instructions will assist the user in setting up and operating the SPEREXSYS. They are incomplete in that a thorough understanding of the concepts and algorithms embodied in the SPEREXSYS are necessary in order to properly exercise the system.

Set Up

Section A of Chapter IV explains the reasons for the following procedures. To set up the SPEREXSYS, the following steps should be taken in the order in which they are listed here. These steps describe how the SPEREXSYS was set up for this thesis research. It can be done other ways using other equipment. All of the equipment used was accessed through the terminals in terminal rooms (room 125), of building 640, WPAFB, Ohio.

1. Turn on the Anderson-Jacobsen 300 baud teletype terminal.

2. The "on line" key should be pushed to the down position.

3. The modem should be set for full duplex.

4. The telephone multiplexer box should have the "b" button pushed in. This connects the telephone to the modem.

5. Dial 4363 or 4362 (access number for the DEC-10) and wait for the computer ready tone. Place the phone in the modem cradle.

6. Hit the <cr> on the terminal until the log in message begins printing.

7. Type: "log 6664,325<cr>". (This is the Milne DEC-10 account number).

8. Type in Milne's password.

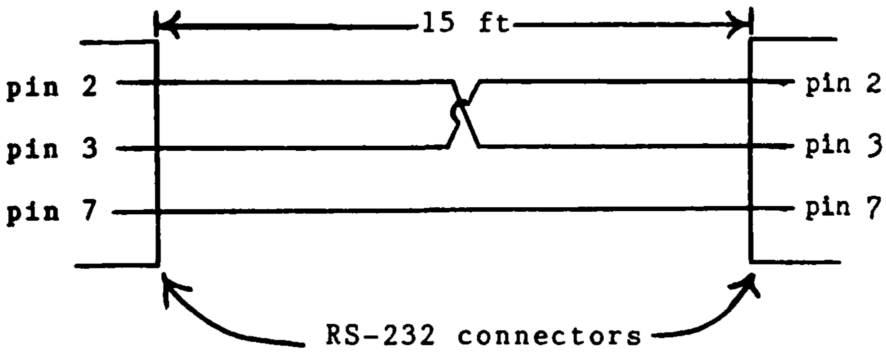
9. Type: "set tty no echo<cr>".

10. Type: "run routh<cr>".

11. Wait until the terminal has printed "yes" and "?-".

12. Unscrew the two screws holding in the RS-232 cable connector on the back of the modem (for the cable which conncects the modem to the terminal).

13. In its place, plug in one end of the specially constructed RS-232 cable. A wiring diagram of this cable is as follows:



14. Select a free terminal with a Gandalf modem

without a phone connected.

15. Turn off the Gandalf modem.

16. Dial 61 on the modem.

17. Turn on the terminal.

18. Press the "set up" key and the reset key.

19. At the sound of the beep, turn on the Gandalf modem.

20. After the request-to-login message has printed on the C.R.T., enter: "rrouth". After the request for password has printed, enter the password.

21. After the system prompt (a percent sign) appears, type: "tty<cr>".

22. Note the response. It will be of the form: "%/dev/ttyxz" (where the xz are variable). This response will be used in step 34.

23. Leave this terminal on and alone. You may wish to post a "do not touch" sign on it.

24. Select another terminal with a Gandalf modem and without a telephone, which is within ten feet of the Anderson-Jacobsen teletype modem.

25. Turn off its Gandalf modem.

26. Dial 60 on the modem.

27. Turn on the terminal.

28. Press the "set up" key and set the transmit and receive speeds to 300 baud. Press the "set up" key again when this has been done.

29. Turn on the modem.

30. Repeat steps 20 - 22 above for this terminal. (Note the response to step 22, it will be used in steps 42 and 47).

31. When the next "%" appears, type: "stty dec<cr>".

32. When the next "%" appears, type: "stty -echo<cr>".

33. When the next "%" appears, type: "lisp<cr>".

34. When the "->" appears, type: "(setq piport (infile '/dev/ttyxz)) <cr>", where the values of xz were obtained from step 22 above.

35. Repeat steps 12 and 13 above (using the other end of the specially built RS-232 cable) for this terminal.

36. Select another terminal with a Gandalf modem and without a telephone.

37. Repeat steps 15 - 20 above for this terminal.

38. After the "%", type: "emacs spxs<cr>".

39. Type: "^s tty<cr> ^s<cr>".

40. Note the tty I.D. (two characters -- the cursor will be bouncing at the first character of the I.D.). Call this I.D. qr for reference in step 42.

41. Type: "<esc> < <esc> q".

42. Type: "ttyxz<cr> ttyqr<cr>", where thevalues of xz were obtained in step 30 above and the values of qr were obtained in step 40 above.

43. Type two spaces.

44. Type "^x^f" and wait for EMACS to exit and the "%" to appear.

45. Type "lisp<cr>".

46. When the "->" appears, type "(load 'spxs)<cr>".

47. When the next ">" appears, type: "/dev/ttyxz<cr>" where the values of xz were obtained in step 30 above.

48. The SPEREXSYS has now been completely set up.

Initialization

The answer to the first question is the value of searchdepth. Enter the value and hit <cr>. The answer to the second question is the value of acceptthresh. Enter the value and hit <cr>. The SPEREXSYS is now initialized.

If the reader does not know how to choose values for searchdepth and acceptthresh, he should reread chapters III and IV of this thesis.

Operation

During the operation of the SPEREXSYS, the user will face two types of questions. The first type of question is regarding the input to the SPEREXSYS from the Voice Decoder. For help here, the user should consult section B of chapter IV and appendix B.

The second type of question which the user will face is the question which follows every complete exchange of stringlist and epreslist between the EPFE and the English Parser. This question serves two purposes. The primary purpose is to prevent faulty input from the Parser to the EPFE (due to line hits or whatever) from causing catastrophic termination of a run. The secondary purpose is to provide the knowledgeable user with the option to reprogram any part of the SPEREXSYS dynamically during operation without stopping the run. This was established primarily as a debugging and evaluation tool but can be used for virtually anything.

This second type of question takes the form:

Would you like to try the EP interface again?
(r-rerun; i-new instrs; g-keep going).

Normally the user will respond with: "g<cr>". This will be used when there is no reason to rerun the EPFE-English Parser data exchange.

If the user has reason to suspect the reliability of the data exchange, he may rerun the complete exchange (for all strings) by typing: "r<cr>".

If the user wishes to reset the value of variables, redefine function definitions, or anything else the Franz LISP language will allow, he should type: "i<cr>". The program options and usage are self explanatory if the "i" option is selected.

APPENDIX D

A DISCUSSION ON HOW A SPEECH RECOGNITION SYSTEM WHICH MODELS THE HSRS SHOULD BE DEVELOPED

D. A Discussion on How a Speech Recognition System Which Models the HSRS Should Be Developed

It is quite evident that the speech recognition problem is a very difficult problem to solve. Millions of dollars have been spent over the last couple of decades in the efforts of some of science's best educated minds using the most advanced research facilities that have ever existed to solve the speech recognition problem. The solutions which have been thus far developed fall so far short of a general solution to the problem that it is quite clear that man has only barely begun in this very difficult trek.

Those conclusions which have been reached so far as a result of the speech recognition research are listed as follows:

1. An extremely accurate (compared to the best that technology has developed so far) acoustic analyzer is required at the front end.

2. The acoustic analyzer must categorize and distinguish sounds the same way that the HSRS does, else it will not be forgiving of the feature measurements which are not very important and it will not be critical enough of the feature measurements which are very important. The result will be an acoustic analyzer which makes different decisions and comes to different conclusions than does the HSRS.

3. A syntactic analyzer must be used which functions around the kernel principle of one word lookahead.

4. Several layers of semantic analysis must be employed because perfect acoustics and perfect syntactics are insufficient to accomplish the task of speech recognition. Some of the levels of semantic analysis have not even yet been clearly identified nor have theories been proposed and developed as to how these levels operate and interface with the rest of the system.

5. All the processing must be done in some manner which allows for parallel or deterministic analysis. There is too much analysis to be done to accomplish the necessary results any other way.

An existence proof that a general English speech recognition system can be, and has been, developed is evident in that humans can communicate in spoken English with each other. Since it is necessary for the speech recognition system to function identically to the HSRS (in order to interpret speech the way the HSRS interprets speech -- a presupposition of the transmission mechanism), then it is obvious that the HSRS is the optimal solution. (Note - perhaps there are more efficient ways of communicating with sound, but none will be more optimal for the task of interpreting human speech than the HSRS).

Ultimately, then, the general solution to the speech recognition problem is to mimic the HSRS. Research which is directed in any other way is doomed to produce less than optimal results. These other research directions may produce solutions which are adequate for some restricted application, but they will not produce a general solution.

To proceed toward the development of this optimal solution by exploring, in detail, the most intricate internal workings of the human brain, and then attempting to reconstruct the entire system from the bottom up, will likely require a very long time to arrive at a solution (if a solution can even be arrived at in this manner). One does not need to know how the electron shells of iron, carbon, hydrogen, and oxygen function in order to design and build an automobile. Indeed, even the most thorough understanding of the electron shells of these atoms will be insufficient knowledge for designing and building an automobile even though one can be constructed based entirely on the functions of the electron shells of these atoms. One needs only to have a rudimentary understanding of the principles of mechanics, thermodynamics, and fluid dynamics along with the wisdom and willingness to experiment with ways to employ these principles together, and one can , in a few years of tinkering, build a car. Henry Ford is an existence proof of this assertion.

The optimal solution to the speech recognition problem must be approached in this manner. Mankind is not going to ever understand how the human brain works by studying neurons. Instead, he must seek to discover the elementary principles of operation (which are implemented by the neurons) of the human brain. Not all of them have to be understood. Just a rudimentary understanding of a few key principles along with the wisdom and willingness to experiment with ways to employ these principles together, and one can, perhaps in a few years of tinkering, build a machine which is functionally similar to the HSRS.

Some of these principles have already been discovered. The one which the remainder of this discussion will center on is the principle which has been suggested by Kabrisky (Ref 11) and explored and verified by several follow-up researchers in the last eighteen years (see the works and bibliographies of Ginsberg). The principle can be stated in a very rudimentary form as follows:

> The human brain interprets and categorizes its sensory input based on a low frequency Fourier transform of the sensory input projected on a two dimensional surface.

Actually, Kabrisky et al have demonstrated this primarily with respect to certain aspects of visual image interpretation. Several observations lead one to suspect that this is true also in the auditory processing portions of the human brain. These are:

1. The physiology of both portions (visual and auditory) of the brain are the same.

2. The physiology of the brain appears to support the

ability to do low frequency Fourier transforms.

3. The auditory processes of the human brain do use Fourier transforms in that it is known that:

a. The auditory nerve carries the first Fourier transform of the sound input to the ear drum.

b. The tonotopic mapping of the auditory cortex in the human brain reveals that the first Fourier of the input sound is mapped onto this two dimensional surface.

A quick review of the brain physiology (Ref 11) reminds us that the image on the retina of the eye is mapped homeomorphically onto the surface of the visual cortex of the brain. If a low frequency Fourier transform of this mapping is compared to the elements in the stored vocabulary, the closest distance match (based on these Fourier features) will produce the correct interpretation of the image. "Correct" in the above sentence means the interpretation which a human would have chosen. This mechanism has been demonstrated to work successfully with not only simple geometric shapes such as letters and numerals, but also with abstract shapes such as photographs of animal cookies. It also is completely sufficient in itself to explain an entire class of optical illusions.

In applying this identically same mechanism to the audio processing portions of the brain, the following mechanism is suggested:

> The brain takes low frequency Fourier transform pictures of the sounds mapped on the auditory cortex in order to interpret and categorize these sounds. In speech, these sound mappings are known as phonemes. It can be psychologically demonstated that phonemes are the elementary acoustic alphabet (symbol set) of speech. If these phonemes are strung together, they will accurately "spell" the words being spoken. If these phoneme strings (which are constructed by time shifting the Fourier transforms of the individual phonemes -- the cepstrums of the sounds) are then compared with words in a stored word dictionary, the closest match should be the same word which the HSRS would have chosen.

In this way, it is possible to build a voice decoder, which satisfies the required function of the voice decoder needed for the front end of the SPEREXSYS, which is as accurate as the acoustic analyzer in the HSRS.

At this point, one wonders how syntactic and semantic analysis is carried out by performing low frequency Fourier transforms of strings of words. This is an unsolved problem.

If the principle of low frequency Fourier image interpretation continues to be the key principle in operation for these levels of analysis, as it has been hypothesized to be in the levels of analysis up to and including word identification, then the mechanism (algorithm) which the SPEREXSYS employs to impose syntactic constraints on the output of the voice decoder is incorrect. This is an area to be explored by future research.

APPENDIX E

THE SHORT TERM MEMORY PHENOMENON IN THE HUMAN SPEECH RECOGNITION SYSTEM

E. The Short Term Memory Phenomenon in the Human Speech Recognition System

According to this researcher's observation of English conversations, there appears to be a facility in the Human Speech Recognition System (HSRS) for choosing the intended word instead of the spoken word in a sentence. An example of this is the fact that almost invariably, a casual human listener will substitute the word "pigs" for the word "picks" in the following spoken sentence: "Farmer Brown had 15 cows, 42 sheep, 7goats, 11 horses, 12 picks, and 29 chickens."

Of course, in this example, the substitution can be credited entirely to semantic analysis. There are, however, other examples when semantics seems insufficient to explain substitutions of this type.

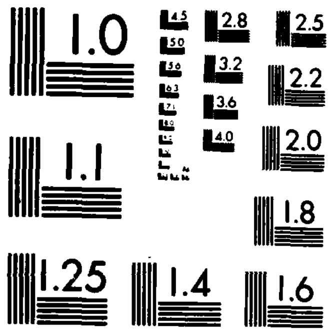
A substitution, by the listener, of the word "pigs" for "picks" is also likely in the following spoken paragraph:

> In the old mining town up in them thar hills about 25 miles due north of Cripple Creek, was an all purpose genral store. Mostly, that thar store sold mostly mining tools and livestock. The miners come down, each man, about ever' six months to buy new tools and livestock. He'd use the tools fer his work in the mines,

AD-A136 835 A SPOKEN ENGLISH RECOGNITION EXPERT SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING R L ROUTH SEP 83 AFIT/GCS/EE/83S-01 3/3

UNCLASSIFIED F/G 9/2 NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

and he'd keep the livestock penned up nearby fer eatin'. Course, thar weren't no money. Ever'body paid in gold dust or gold nuggets. If a miner'd done real good the last six months or so, he'd maybe buy a shovel, if he broke his, and a cow and maybe a couple pigs. Some miners jest buyed shovels and stuff. Some 'ud jest buy pigs to eat cause they didn't break no tools. One year, I jest bought two picks.

Semantically, either word "picks" or "pigs" would fit as the last word in the above paragraph. The word "picks" was actually uttered, but the word "pigs" will likely be substituted by the HSRS. The listener may, instead of making the decision to substitute the word, simply ask for clarification with a question such as: "Did you say 'two picks' or 'two pigs'?" The fact that the last word's identity is not clear to the listener even though it was distinctly uttered as "picks" is an indication that the HSRS prefers words which have recently been uttered over new (as yet unuttered) words.

There is ample evidence to suggest that in human visual image interpretation, a short term memory is actively involved in the brain's image interpretation. It serves, among other functions, to focus future image interpretation on recently (past) interpreted images. It also fades very quickly in a manner which appears to be related to the logarithm of time since the image was projected on the visual cortex.

It seems reasonable to assume that some similar type of short term memory is active in the HSRS. That would be sufficient to explain the substitution of the word "pigs" for "picks" in the last example.

This is a phenomenon which deserves far more research and experimentation than has been done for this thesis. This researcher is not willing to state conclusively that such a phenomenon exists or that if it does exist, it does influence speech recognition in the manner described.

This researcher is saying, however, that there appears to be evidence which suggests that a short term memory is operational in the HSRS. For this reason, a crude short term memory has been modeled into the SPEREXSYS. The results of test number two in chapter IV suggest that this addition was useful in improving the performance of the SPEREXSYS.

APPENDIX F

THE PHENOMENON OF FAVORING LONGER WORDS IN THE HUMAN SPEECH RECOGNITION SYSTEM

F. The Phenomenon of Favoring Longer Words in the Human Speech Recognition System

It was hypothesized by this researcher, as a result of observing normal human speech communication, that when the HSRS is given a choice between interpreting an input utterance as two short words or one long word, it will almost always choose the long word. It was further hypothesized that the decision to choose the longer single word over the two shorter words would always be made unless the syntactic or semantic analysis levels influenced the decision to the contrary. For example, if syntax and semantics are not involved, and the three words "come and ding" are spoken in connected speech (so that "and" and "ding" share the "d" phoneme), the HSRS will prefer to interpret the utterance as the single word "commanding" rather than "command ding" or "come and ding." Each of the three options could plausibly fit both syntactically and semantically in an English sentence.

To test this hypothesis, a series of informal experiments were performed. The students of a graduate level course on pattern recognition participated as the subjects of the experiments. The experiments were administered on two different class days. The subjects were not an ideal group to participate in the experiments, because they all tended to be of more than average intelligence and had all spent over three months, at the time of these experiments, studying the peculiarities of pattern recognition in vision and speech. This is to say that they were far from an unbiased and naive group. Nevertheless, they had graciously volunteered to participate in the experiment and this researcher was grateful for their cooperation as no other subject group was available.

On the first day of the experiments, a female volunteer (not a class member), who was not informed of the purpose of the experiment, appeared before the class to speak the following four words in a manner in which they would be spoken in rapid connected speech: "mass tree toga oats."

Before her utterance of these four words, the class was instructed that they would hear the utterance only once and that immediately following the utterance they were to write down their best representation of the utterance using English words. If they could not find suitable English words, they were instructed to write the word "noise." (Note - The class was not informed as to the purpose or expected results of the experiment).

No context was originally given. Eighteen students participated in the experiment. Their responses to the first utterance of "mass tree toga oats" are as follows:

1. mastery toe goats
2. mastery toe goats

3. master eat old goats
4. mastery tow goats
5. mastery toga
6. master eat toe goats
7. mastery to goats
8. master eat toe goats
9. mastery toe goes
10. mastering toe goes
11. mastery toe goats
12. mastery go goes
13. mastery to go oats
14. noise
15. noise
16. mastery took its
17. mastery to ghost
18. master noise

There are several observations which can be made about the above responses. One of them is that 13 of the 16 participants who responded with something other than "noise" chose to make a single longer word out of "mass tree" than represent the utterance as two words. The other three respondents chose to place the word boundary beyond the intended word boundary so as to make a longer first word than "mass."

These four words were uttered seven more times. Each time the participants were given conversational contextual information in an attempt to introduce semantic influence on the results. Because that portion of the experiment does not have much to do with the phenomenon of choosing single longer words over multiple shorter words, the results will not be presented here. It is significant to say that with proper contextual preparation, all 18 participants were interpreting the first two words as "mass tree" by the sixth utterance. This is to say that semantic analysis can override the tendency to prefer longer word interpretations over shorter word interpretations.

At the end of the experiment on the first day, the purpose of the experiment was explained to the class. They were now a definitely biased group.

On the second day, a second experiment was performed. A set of words was spoken. The participants were asked to write down one, two, three, or four English words which best represented what they heard. Some of the results obtained reflected an obvious attempt on the part of the participant to separate utterances into as many words as possible. Again, this was due to the fact that the participants understood the purpose of the experiment and were, therefore, consciously looking for shorter words which composed, or were close to, the longer words. In spite of this prejudice, the results still indicated that the participants favored longer words. Some examples are presented below.

When the utterance was : "come and ding," the responses were:

1. commanding
2. commanding
3. commanding
4. commanding
5. commanding
6. commanding
7. commanding
8. commanding
9. commanding
10. commanding
11. commanding
12. commanding
13. commanding
14. commanding
15. commanding
16. commanding
17. commanding
18. commanding

When the utterance was: "gun shoot her" (with particular care being given to the pronunciation of the "h" in "her"), the responses were:

1. gun shooter

2. gun shooter

3. gun shooter

4. gun shooter

5. gunshooter

6. gunshooter

7. gunshooter

8. gun shooter

9. gun shooter

10. gun shooter

11. gun shooter

12. gun shooter

13. gun shooter

14. gun shooter

15. gun shooter

16. gun shooter

17. gunshooter

18. gun shooter

It is interesting to note that there was such a pronounced tendancy to combine "shoot" and "her" into the single word "shooter" that in all 18 instances, the HSRS's preferred to ignore the "h" sound in "her" in order to combine the two words into one.

When the utterance was: "sum or seas," the responses were:

1. some mercies
2. sub mercies
3. some mercies
4. some mercies
5. some mercies
6. sum ercies
7. some mercies
8. some mercies
9. some mercies
10. some mercies
11. sum mercies
12. some mercies
13. some mercies
14. some mercies
15. some mercies
16. some mercies
17. some mercys
18. so mercies

Two things are surprising here. The first is that this researcher expected to see the response "summer seas" as the more frequent response. It was not a response at all. The second noteworthy observation is that the "o" in "or" does not sound much like the "e" in "mercies." Nevertheless, the tendency to combine two words into one was overwhelming.

Needless to say, some responses were not quite as convincing of this theory. One of these is as follows. When the utterance was: "pair or shoot," the responses were:

1. pair or shoot
2. parachute
3. parachute
4. pair a shoot
5. parachute
6. parachute
7. pair oh shoot
8. par oh chute
9. parachute
10. parachute
11. pare or shoot
12. pair oh shoot
13. parachute
14. parachute
15. parachute
16. parachute
17. parachute
18. parachute

This is not as was expected in that the response "parachute" was expected more often. Some of the interpretations seemed strained to avoid writing one long word (particularly response number eight).

Overall, the evidence seems to support the hypothesis that longer words are preferred over multiple shorter words by the Human Speech Recognition System.

APPENDIX G

DATA DICTIONARY

Data Dictionary

This data dictionary includes data descriptions and a brief glossary of the explanation of the acronyms used for module names. For the most part, only the global data elements are included, although a few key local data elements are also defined.

The purpose of this dictionary is to assist the reader in understanding both the narrative in chapter III and the LISP program listing in appendix A.

This dictionary is ordered alphabetically by the variable names of the data elements. The module name glossary is at the end of the appendix.

acceptthresh:

Stands for: acceptance threshold

Aliases: m (in EPFE and GLOBAL as a local variable)

Composition: one real number value between zero and one.

Notes:

1. The value of acceptthresh is the minimum allowable average probability of the last three words in a string.

declist:

Stands for: decision list

Aliases: none

Composition: list of dictionary entries,
e.g. - (peak peek peaking)

Notes:

1. This is the list of all third words back from the end of all active strings.
2. This list of words marks the words which the semantic analyzer must comment on (when one has been developed).

<u>dict.name:</u>

Stands for: dictionary entry name

Aliases: dictname, word.dict, worddict

Composition: one of the allowable 200 dictionary entries.
e.g. - the

Notes:

1. Dict.name is a single English word.
2. This is different from the SPEREXSYS definition of a "word".

epoutport:

Stands for: English Parser output port

Aliases: none

Composition: in form: /dev/ttyxz

where xz specify a particular VAX output port

e.g. - /dev/ttyi2

Notes:

1. Used by the SPEREXSYS to redirect output to the DEC-10 on which the English Parser runs.

epres:

Stands for: English Parser response

Aliases: none

Composition: (stringum (feature set) (feature set)...)

e.g. - (1001 (noun) (verb & not adj))

epreslst:

Stands for: English Parser Response List

Aliases: none

Composition: (epres epres epres ...)

e.g. - ((1001 (noun)(verb)) (1002 (adj)(fpunc))) featureset:

Stands for: set of legal features. A feature is a grammatical type.

Aliases: none

Composition: (feature feature feature ...)

e.g. - (all fpunct fpunc noun nlp ...)

Notes:

1. See DICT.SPXS program listing in appendix A for complete definition.
2. Each feature is also defined as a set of dict.names which have that feature as a syntactic function.

init:

Stands for: initial cycle through EPFE

Aliases: none

Composition: variable with the value nil or anything else.

Notes:

1. When not nil, it signifies the EPFE is in its initial cycle for a new conversation.

inittim:

Stands for: initial time

Aliases: inittime and tim (both in EPFE).

Composition: a single integer value.

Notes:

1. This is the time at which the next sentence is expected to start.
2. It is the time the last sentence ended (including FPUNCT -- if there was one).

maxstnum:

Stands for: maximum string number used

Aliases: none

Composition: a single integer value.

Notes:

1. The current value of maxstnum is the most recently (and highest value) assigned string number.
2. Maxstnum+1 will be the value of the next assigned string number.

maxwordtim:

Stands for: maximum word time of utterance

Aliases: none

Composition: single integer value
(currently assigned as 200).

Notes:

1. Time (in Seelandt time units) it takes to pronounce the longest possible word in the vocabulary.

minaccept:

Stands for: minimum acceptance threshold

Aliases: none

Composition: single real number with a value between zero and three.

Notes:

1. Defined as three times acceptthresh.

notfeatset:

Stands for: not in feature set

Aliases: none

Composition: (feature feature feature)

e.g. - (qp np ap pp vp ...)

Notes:

1. See DICT.SPXS program listing in appendix A for complete defintion.
2. Set of features in English Parser which the SPEREXSYS does not recognize.

<u>nextguess:</u>

Stands for: next guess request

Aliases: nxgs, nxguess, nextgs

Composition: (stringnum timl (feature set)(feature set)...)

e.g. - (1001 15 (noun)(verb & not npl))

Notes:

1. This is the basic request for information (next word guesses) which the EPFE sends to the Voice Decoder.
2. One for each active string is sent.
3. They are all sent together as a list of nextguesses. See next dictionary entry.

4. Timl is the approximate start time for the next word to be guessed for that string.

<u>nxgslst:</u>

Stands for: next guess list

Aliases: next-guess-list

Composition: (nextguess nextguess nextguess ...)

e.g. - ((1001 15 (noun)(adj))(1002 20 (verb)(ns)))

Notes:

1. This is a list of nextguess requests.
2. See nextguess dictionary entry above.

<u>numstring:</u>

Stands for: number of active strings allowed

Aliases: none

Composition: a single integer value

(normally searchdepth squared)

Notes:

1. This value determines the number of words lookahead that will be used.
2. Currently it is set to searchdepth squared

which allows for a two word lookahead.

3. It is set in GLOBAL.

<u>optseslst:</u>

Stands for: optional sentence string list

Aliases: none but is used like sentstlst.

See: dictionary entry for sentstlst.

<u>optstg:</u>

Stands for: optional string of English Parser Responses

Aliases: none but is used like epreslst.

See: dictionary entry for epreslst.

<u>optstglst:</u>

Stands for: optional string list

Aliases: none

Use: functions as temporary dummy variable for a variety of other lists.

<u>prob:</u>

Stands for: probability of likelihood of word

Aliases: none

Composition: a single valued real variable.

Notes:

1. This is carried as the last data element of every word and represents that word's probability of likelihood of being the correct word.
2. See dictionary entry for "word".

<u>searchdepth:</u>

Aliases: topchoicenum (in EPFE)

Composition: a single integer valued variable.

Notes:

1. This represents the number of words deep the SPEREXSYS may have to search in the list of next word guesses (for each string) in order to find the correct word. It is therefore an indicator of the reliability of the Voice Decoder.
2. If searchdepth needs to be any higher than three, the SPEREXSYS will not be expected to perform very well.

sentstlst:

Stands for: sentence string list

Aliases: none

Composition: (sentence sentence sentence ...)

e.g. - ((1011 (he (0 15) .98)(went (15 30) .87))
(1012 (she (0 15) .92)(lent (15 30) .81)))

Notes:

1. A sentence has the same form as a string, only it is a complete sentence.
2. This is the global variable that the EPFE builds for the Semantic Analyzer. It contains all the sentences the EPFE has built that are the possible identities of the correct sentence.

shortermem:

Stands for: short term memory words

Aliases: none

Composition: (name.dict name.dict name.dict ...)

e.g. - (the boy hit the girl a big...)

Notes:

1. This is the short term memory's list of remembered vocabulary words.

stringlist:

Aliases: stglst

Composition: ((stringnum word word word) ...)

e.g. - ((1001 (the (0 15) .95)(big (15 35) .90))
(1002 (the (0 15) .95)(pig (15 35) .89)))

Notes:

1. This is the list of active strings

stringnum:

Stands for: string number

Aliases: stgnum

Composition: a single integer valued variable

Notes:

1. Stringnum is the identification tag that marks each different string.
2. No two strings have the same string number.

tim1:

Stands for: time of start of word

Aliases: none

Composition: single real (normally integer) value

Notes:

1. Marks the time a word begins in the input utterance if tim1 is sent from the Voice Decoder.
2. Marks the approximate start time of the next word to be guessed if tim1 is sent from the EPFE.

<u>tim2:</u>

Stands for: time of end of word

Aliases: none

Composition: single real (normally integer) value

Notes:

1. Marks the time a word ends in the input utterance.

<u>topwordlst:</u>

Stands for: top words list

Aliases: none

Composition: ((stringnum word word)(stringnum word word word) ...)

e.g. - ((1001 (the (0 15) .95)(a (0 15) .90))
(1002 (big (0 15) .92)(door (0 15) .86)))

Notes:

1. List of the top (most probable) searchdepth number of words for each string.

word:

Aliases: none

Composition: (name.dict (tim1 tim2) prob)

e.g. - (the (0 15) .95)

Notes:

1. Different from name.dict.

wordgslst:

Stands for: word guess list

Aliases: none

Composition: (wordgs wordgs wordgs ...)

In form it is identical to topwordlst.

Notes:

1. This is the Voice Decoder's response (of next-word-guesses) to the EPFE's next-guess-request.

<u>words:</u>

Aliases: none

Composition: (word word word ...)

e.g. - ((the (0 15) .95)(big (15 30) .92))

Notes:

1. The same as a string with the string number removed.

Glossary of Module Name Acronyms

ADDTOSESTLS -- Add to Sentence String List

AUGSENSTG -- Augment Sentence String

BUILDDECLST -- Build the Decision List

BUILDNXGS -- Build Next Guess Request

CALCNEWPROB -- Calculate New Probability

CALCSTGPROB -- Calculate the String Probability

CHANGEPROB -- Change the Probability

CHECKFPUN -- Check for Final Punctuation (FPUNCT)

CHECKMINPR -- Check Minimum Probability

CHOPTOMNS -- Chop to Minimum Number of Strings

COMPLIMENT -- Compliment

DECTOPWDS -- Decide on Top Probability Words

ELIM -- Eliminate

ELIMINACC -- Eliminate (all strings below the) Minimum Acceptable String Probability

EPFE -- English Parser Front End

ERRORRECOVRY -- Error Recovery

EVALS -- Evaluate the Instruction

EXAMNEXT2 -- Examine Next2

FINDFPUNCT -- Find Final Punctuation (FPUNCT)

FINDTIM1 -- Find Tim1 (word start time)

FINDTOPWDS -- Find Top Probability Words

FINDWDSMATCH -- Find Words Match

FORMNXGS -- Form Next Guess Request

FPUNCTPROC -- Final Punctuation (FPUNCT) Processor

GETNEXT1 -- Get Next1

GETPROBLST -- Get List of Probabilities

GETSTGPROBS -- Get String Probabilities

GETTIM1 -- Get Tim1 (word start time)

GETTOPSTS -- Get Top Probability Strings

GETTOPWDS -- Get Top Probability Words

GETWORDOPTS -- Get (next) Word Options

GETWORDS -- Get Words

GLOBAL -- Global Initialization

INCRFPUNCTS -- Increment FPUNCT Probabilities

INTERFEP -- Interface with English Parser

INTERFVOCDEC -- Interface with Voice Decoder

INTERSECT2 -- Intersect Two Sets (lists)

ITEPREST -- Iteratively Strip English Parser Responses

KILLOWSTS -- Kill Low Probability Strings

KILNILSTS -- Kill Nil Strings

LOOKATNEXT2 -- Look at Next2

MAKEDECISION -- Make Decision

MAKENXGSLST -- Make Next Guess Requests List

MAKESTS -- Make Strings

MODFPUNCTS -- Modify FPUNCTs

NEWSESLST -- New Sentence String List

NEWSTRINGS -- New Strings

ORDERLIST -- Order List

ORDERSENTLST -- Order Sentence String List

OUTRESTSENT -- Output the Rest of the Sentence

OVERMIN -- Over Minimum

PROCFEATTERM -- Process Feature Term

PROCWDGS -- Process Word Guess Response

PRINTSENT -- Print Sentence

PRINTWORDOPTS -- Print (next) Word Options

RANKSENTS -- Rank Order Sentences (by probabilities)

REINIT -- Re-initialize

SEMANALYZER -- Semantic Analyzer

SEMANINIT -- Semantic Analyzer Initialization

SHORTERMEMPROB -- Short Term Memory Probability

SHORTTERMMEM -- Short Term Memory

SPEREXSYS -- Spoken English Recognition Expert System

SPXSINIT -- SPEREXSYS Initialization

STARTNSTS -- Start New Strings

STGPRINT -- String Print

STGPROB -- String Probability

STRPTOPN -- Strip Top "N" Number (off of list into new list)

TOPFPUNCT -- Top Final Punctuation (FPUNCT)

TOPSENT -- Top Probability Sentence

TRANSLATE -- Translate

UNION2 -- Form the Union of Two Sets (lists)

USERFDBK -- User Feedback

VITA

Education

Richard LeRoy Routh graduated from the United States Military Academy in 1978 with the degree of Bachelor of Science with areas of concentration in Nuclear Engineering and Computer Science. In 1978, he also graduated from the U.S. Army Signal Officers Basic Course. In 1979, he graduated from the Radio Systems Officers Course at Fort Gordon, Georgia. In 1982, he graduated from the University of Phoenix with the degree of Master of Arts in Management. In 1983, he graduated from the U.S. Army Signal Officers Advanced Course and the Air Force Institute of Technology Computer Systems Training program.

Assignments

Since his graduation and commissioning from the United States Military Academy, his assignments have included: ADP Officer in the Telecommunications-Automation Directorate of the U.S. Army Communications and Electronics Engineering and Installation Agency (USACEEIA-TAD); Aide-de-Camp to the Deputy Commanding General of the Army Communications Command; Platoon Leader and Executive Officer of A Company, 40th Signal Battalion, Fort Huachuca, Arizona; and Operations Officer for the 86th Signal Battalion, 11th

Signal Brigade, Fort Huachuca, Arizona.

Publications and Honors

In 1977, he submitted a paper titled, "A Pulsed, Nitrogen Gas Laser," to the Eastern Colleges Science Conference and was given the second place award in engineering for this work. He published an article titled, "Computing Optimum Repair Parts Inventory Cost," in the Army Logistician magazine, January - February 1981. His Master of Arts in Management thesis was titled, "Basic Principles and Guidelines for the Successful Management of a Small Business," and was published with the University of Phoenix, Phoenix, Arizona.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFIT/GCS/EE/83S-01	2. GOVT ACCESSION NO. AD-A136 135	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Spoken English Recognition Expert System		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard LeRoy Routh		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE September 1983
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

AFR 190-17. 9 SEP 1983

Dean ... and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

- Speech recognition
- Speech analysis
- Syntax

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Subject to the accuracy of the acoustic analyzer and the accuracy and completeness of the English Parser, a real-time general solution to the application of English syntactic constraints to spoken to Spoken English recognition has been developed. This solution is functionally equivelent in many ways to the syntax processing of speech in the human brain. Because it closely models the syntax processing of the Human Speech Recognition System (HSRS), it is most effective when used with the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) item 20 continued- several levels of semantic analysis which are also evidently operational in the HSRS as has been shown in this thesis. Hence, this work may well be a necessary part of the eventual general solution to the speech recognition problem.

END

FILMED

2-84

DTIC